月刊マイコン別冊

X68000 活用研究 II

X-BASIC マスター編

© SEGA 1985

宮原哲也/深沢幸三共著

SHARP

ブラックタイプ新登場

通商産業省選定
グッドデザイン商品

もっと先の話をしよう。

クリエイティブワークステーションX68000。

分野を問わず、既存にこだわらないものを創り出すことはたいへんな苦労をともなうものです。傑出した創意と情熱、そのプロダクツに対する将来的な展望。机上での設計は、なるほど簡単かも知れませんが、それを世に問う場合の責任の重大さは並々ならぬものです。とりわけパーソナルコンピュータの分野では、必然的にソフトウェアの資産が問われ、ハードウェアが一人歩きすることなど、かなわないのが現状でした。今、さかんにとりざたされている、いわゆるコンパチブル路線も、まさにそうした市場環境が生み出した産物でしょう。

X68000が登場して八ヶ月、ソフトウェア面ではほぼ100%白紙の状態で世に問わざるを得なかったこのマシンが、これほどまでに熱いご支持をいただいたことに、ユーザー各位に心から感謝するとともに、開発当初より5年先を見つめてきたその思想に意を強くするものです。そして今、このマシンのポテンシャルにふさわしいソフトウェアの登場で、また新たな局面を迎えようとしています。次のステップへ、X68000はさらに飛躍してゆきます。

●実装密度を追求したフォルムー新のマンハッタンシェイプ●広くリニアなアドレス空間、68000搭載●テキスト、グラフィック、スプライト、独立3画面設計、2Mバイトの大容量メモリ●フレンドリーOS、Human 68k搭載●連文節変換、マルチフォントをサポートした強力日本語処理●1024×1024ドット(最大表示エリア768×512ドット)の実画面エリアを装備した高解像度表示能力●512×512ドット、65,536色同時発色●水平32、1画面128のスプライト機能●オーバースキャン機能を採用した512×512ドットレベルのスーパーインポーズ●テキストビットマップ方式採用●8重和音ステレオFM音源搭載●音声デジタイズ記憶AD PCM●新開発マウス・トラックボール●1Mバイト5"FDD 2基搭載●X-BASIC、日本語ワードプロセッサ、グラディウス同梱

豊富な周辺機器がクリエイティブワークをサポート。

品名	型番	価格
●15型カラーディスプレイ	CU-15M1(E・B)	標準価格 99,800円
●カラーイメージユニット	CZ-6VT1	標準価格 69,800円
●カラービデオプリンタ	CZ-6PV1	標準価格198,000円
●24ピン漢字プリンタ(80桁)	CZ-8PK7	標準価格122,000円
●24ピン漢字プリンタ(136桁)	CZ-8PK8	標準価格152,000円
●24ピン漢字プリンタ(80桁)	CZ-8PK9	標準価格 89,800円
●熱転写カラー漢字プリンタ	CZ-8PC2	標準価格 69,800円
●ハードディスクユニット(10MB)	CZ-500H	標準価格348,000円
●増設用ハードディスクユニット(10MB)	CZ-501H	標準価格258,000円
●ハードディスクユニット(20MB)	CZ-620H	標準価格178,000円
●モデムユニット	CZ-8TM2	標準価格 49,800円
●RS-232Cケーブル(平行接続型)	CZ-8LM1	標準価格 7,200円
●RS-232Cケーブル(クロス接続型)	CZ-8LM2	標準価格 7,200円
●1MB増設RAMボード(内蔵用)	CZ-6BE1	標準価格 35,000円
●拡張I/Oボックス	CZ-6EB1	標準価格 88,000円
●2MB増設RAMボード*	CZ-6BE2	標準価格 79,800円
●4MB増設RAMボード*	CZ-6BE4	標準価格138,000円
●GP-IBボード	CZ-6BG1	標準価格 59,800円
●ユニバーサルI/Oボード	CZ-6BU1	標準価格 39,800円
●増設用RS-232Cボード(2チャンネル)	CZ-6BF1	標準価格 49,800円
●数値演算プロセッサボード	CZ-6BP1	標準価格 79,800円
●アンプ内蔵スピーカーシステム(2本1組)	AN-160SP	標準価格 59,800円
●ジョイカード	CZ-8NJ1	標準価格 1,700円

*ご使用の際にはCZ-6BE1が必要です。

パーソナルワークステーション

X68000

●本体+キーボードCZ-600C(E・B) 標準価格 369,000円

●15型カラーディスプレイテレビCZ-600D(E・B) 標準価格 129,800円

●チルトスタンドCZ-6ST1(E・B) 標準価格 5,800円

●拡張I/OボックスCZ-6EB1 標準価格 88,000円

シャープ株式会社

●お問い合わせは…シャープ(株)電子機器事業本部システム機器営業部〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)
電子機器事業本部テレビ事業部第4商品企画部〒162 東京都新宿区市谷八幡町8番地☎(03)260-1161(大代表)へ。

READY? 「準備はととの

MANY MORE BA
WILL SOON BE

TOP 1000000 SCORE 9720
STAGE 1

SQUILLA

TOP 1000000 SCORE 790810
PUSH ENTER OR ESC
STAGE 4

BINSBEEN

TETRA

TOP 1000000 SCORE
PUSH ENTER OR ESC

GODARNI

DOM

TOP 1000000 SCORE

SYURA

TOP 5708150 SCORE
PUSH ENTER OR ESC

VALDA

TOP 1000000 SCORE

ROLLYS

TOP 1000000 SCORE

MANMOS

Story：銀河系の遥かかなた、ここドラゴンランドは、平和と愛に満ちあふれた、地球型星系であった。だが突如発生した超自然現象と正体不明の凶悪な魔生物の出現により、その様相はすっかり変わってしまったのである。地球人の超能力戦士“ハリアー”に、正義のドラゴン“ユーライア”から救助を求める連絡が入ったのは当然のことだったと言えるだろう。「コノ　セカイノ　ヘイワヲ　トリモドシタイノダ！　イソイデ　クレ！　ハリアー！」ユーライアの悲痛な要請を受け、ハリアーは、ついに起った。愛用のオートロック（自動照準固定）機能装備のランチャーを携え、ハリアーは前進を開始した。
走れ！翔べ！撃て！スペースハリアー！

SPA
HARI

マイコンソフト
電波新聞社 出版販売部
〒141 東京都品川区東五反田1丁目11番15号
電話(03)445-6111

たかい? 戦闘は間もなくはじまるぜ!」

TTLE SCENES AVAILABLE.

URIAH

OCUTOPUS

SALPEDON

MUKADENS

BARBARIAN

STANLAY

JET

WIWIJAMBO

TITLE

ACE RIER

© SEGA 1985

お待たせしました。X68000用スペースハリアーがついに発売になりました。多重画面、スプライト、FM音源、高速演算処理、多色発色など、X68Kの持つすぐれた能力を活かし、68000×2、Z80×1という重装備を誇る、アーケード・マシンのオリジナル・ボードの表現力に肉薄した完成度で皆様のお手元にお届けすることができました。
店頭デモでPLAYする場合にも、ヘッドホンをお使いいただき、Hi-Fiサウンドをバックにお楽しみください。

スペースハリアー for X68000

5インチ Disk版 標準価格6,800円

マイコンソフト

DRAGON BUSTER

The soldier Clovis went to Mt.Dragon, running after Dragon. Dragon had taken Princess Celia as a hostage. But, monsters in the graveyard and ruins interrupted his way.

セリア王女がドラゴンにとらわれた！
王女の身とローレンス王国を救うために立ち上がったのは、
ドラゴンバスターとしての力を内に秘めたクロービスであった。
待ち受けるルームガーダーを倒し、アイテムを集めながら
最終ラウンドに潜むドラゴンを打ちやぶるのが使命だ。
ユメとロマンにあふれた冒険が今はじまる。

ドラゴンバスター

シャープX1 5インチFD版
標準価格6,200円

©株式会社 ナムコ

Presented by

DEMPA MICROCOMPUTER SOFTWARE

©株式会社ナムコ 発売元：電波新聞社 〒141 東京都品川区東五反田1-11-15 ☎03-445-6111

はじめに

X68000発表から１年が過ぎました。あの発表の時には夢であったものが，今，皆さんの手元にあります。何をしていますか？

〝グラディウスを克服し，今スペースハリアーで10面で悪戦苦闘〟なんていう人もいるでしょう。〝早くOS-９／68000が出ないかなぁ〟と本体を前にして指をくわえている人も…。

X68000は欲ばりなほど，いろいろなおまけが付いてきます。標準でOS（Human68k）とワードプロセッサ，辞書，ゲーム〝グラディウス〟が付いています。このおまけ，皆さんがもしかすると一生のお付き合いになるかもしれません。ゲーム〝グラディウス〟で一生…？　なんて。ワープロで一生…？　これは，考えられないこともありませんが…。OSのディスケットの中に，BASICというフォルダーが目につきます。BASICなら，一生のお付き合いもできそうです。〝BASIC〟あまりにも耳になじみのある言葉です。

〝Beginners Allpurpose Symbolic Instruction Code〟私には，もう10年の付き合いになります。これからもずっと一緒に付き合っていくことになるでしょう。

皆さんも一通りOS（Human68k）の操作がわかったところで，このBASICというフォルダーを開いてみたことと思います。いや，その前に福袋の中にある〝X68000のテーマ〟を聞いていると思います。あなたのX-BASICとの付き合いは，その時にもう始まったのです。あの，何ともリズミカルな演奏。あの，自動演奏プログラムこそ，X68000付属のX-BASICによるものなのです。

OSはそのマシンの中心になる基本的な存在ですが，OS自身では何の創造もできません。そのOS上にあるプログラミング言語こそが，私たちに創造への門を開いているのです。プログラムを創造するよろこびこそが，パーソナルコンピュータに残されたユーザーの特権なのです。

この本を書くにあたりわれわれCZ masters clubでは，「プログラミングとは何か」を自問自答してみました。個々によって，考えはまちまちになったことは明白ですが，基本となる考え方は同じでした。創造へのおしみない努力と探究心。これがすべてを解くカギとなっているのです。われわれユーザーは，このX68000から見ればまだ赤ん坊です。常に「これは何？」，「これは，どうして？」と問い続けています。疑問を持ち，とことんまで追求して失敗をかさねる。そんな中から少しずつ理解し成長していくのです。

本を書いた側と読む側でのギャップは，ほとんどないと信じています。X68000については，皆，同じスタート地点に立っているのですから。X-BASICについて，ほんの少しみなさんより早く覚えたというだけのことなのです。そんなことは，X68000の宇宙の広さからすれば，ほんのちっぽけな塵みたいなものなのです。

これからの努力と探究心で，皆さんと共にX68000の宇宙を探検し少しずつこの宇宙地図を埋めていこうではありませんか!!　旅はまだこれからです。

昭和62年11月７日

CZ masters club　宮原哲也
深沢幸三

ライトスタッフ　佐野勝栄
鎌田由香

X68000活用研究 II
X-BASICマスター編

第1章 X-BASICの概要

第2章 X-BASICの基礎

CONTENTS

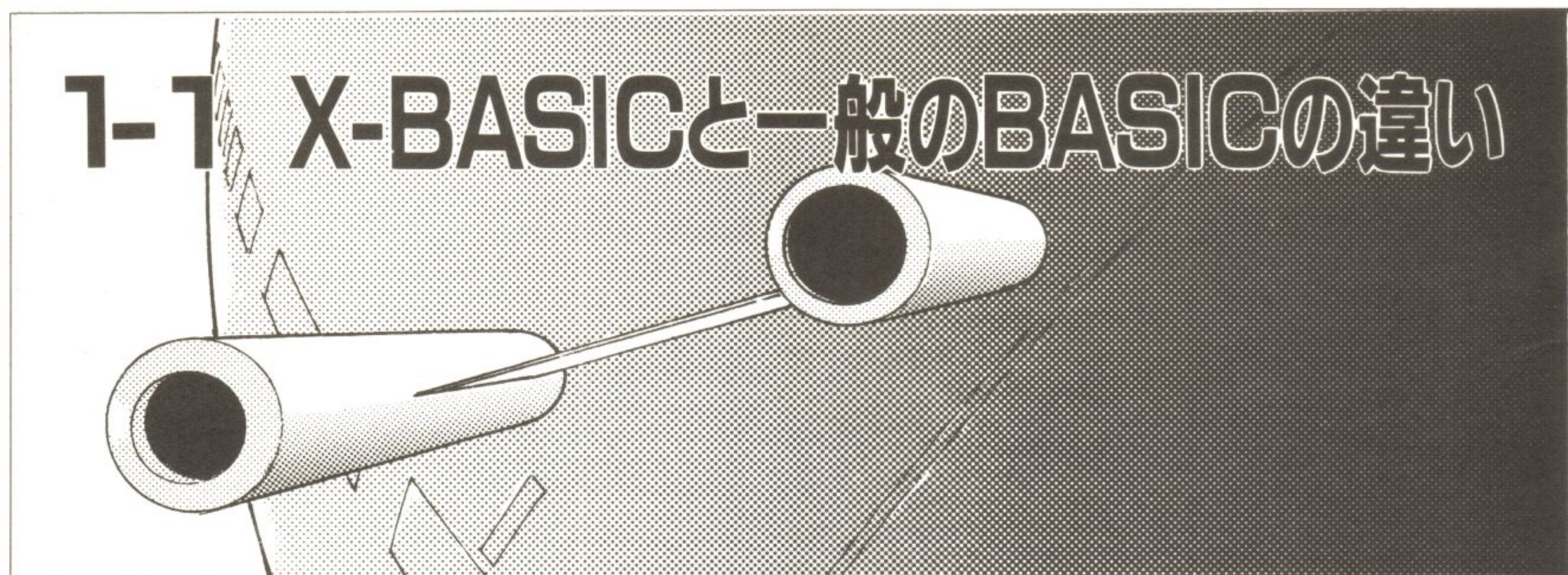
第1章

X-BASICの概要

1-1　X-BASICと一般のBASICの違い

1-2　X-BASICの特長

1-3　これからのX-BASIC

1-1 X-BASICと一般のBASICの違い

X68000はこれまでのマシンとは比べものにならない程ハードウエアの内容が充実していて，その集積度は現在の個人レベルで取得可能なパーソナルコンピュータの限界ではないかと思われる程です。コンピュータのハードウエアとソフトウエアは調和が取れていなければなりません。どんな良いソフトもハードウエアが貧弱だとその良さが発揮出来ないし，またその反対にハードウエアが良くてもソフトウエアの出来が悪いと全くハードウエアの良さが出てきません。そんな時勢に，シャープはX68000にオリジナルOS Human68kをぶつけてきました。このOSはビジュアルシェルを使って視覚に訴える国産では目新しい物です。このOSにはこれで全く問題がない訳ではありませんが，一つの新しい国産機の方向ずけになっています。

X-BASICはこんなOS上で走るインタープリンタBASICなのです。このBASICは非常に構造化されたプログラムを記述することができ，今までのBASICとは一味違ったものに仕上がっています。『構造化されたプログラム』と言ってもピンとこない方も多いかと思いますが，一言で言えば，『わかりやすいプログラム』と言えます。プログラムを作っている時や，デバックの時に『わかりやすいプログラム』が力強い味方になると言う事です。

例えば，今までのBASICの場合処理速度を上げるためによく使われるテクニックで，サブルーチン(繰り返し実行されるような処理をどこからでも使える様にパックしたもの)をプログラムの先頭近くに持って来ます。この場合メインルーチンは，これらのサブルーチンを避けて後ろの方へ置かなければならなくなります。こんな時は，ごく普通にプログラムの先頭にgoto～と記述しメインルーチンを後ろへ持っていきます。これではプログラムの解読に非常に手間が掛かりますし他人のプログラムに至っては解読など神技で，とても解読できたものではありません。

また，プログラムと言うものはどんどん育っていくもので，処理の充実を計る時や，デバッグする時などは行と行の間に別の処理や何やらがごちゃごちゃと入ってきます。そんな時におおよそのプログラマーは，広いスペース（空き行）を求めて，goto～と処理を移して広い空間でのびのびと色々な処理を加えて元の位置へgoto～と何も無かったかの様に戻ってきます。ちょっと誇張した感じで書きましたが，多かれ少かれそれに近い事は行われているのです。こうなればもう鉄壁のプロテクトの掛かったプログラムで，かなりの根気が無ければリストを見る気にもなりません。とにかく行番号に管理されているプログラムの場合は，打ち込んでいくうちに，思うまま，気の向くままに行番号の空いている所へ跳んでいってしまいます。今までのBASICでは，このgoto～，gosub～returnの嵐の中で，自分でも何が何だか判らなくなってしまうようなプログラムリストに

埋もれていました。

さてこれくらい普通のBASICの事をひどく書いてしまったので，X68000のX-BASICはさぞかし優れ物ではないかと期待されていることでしょう。そうなのです，goto～，gosub～return無しでプログラムを作ることが出来るのですから・・・（しかしながらX-BASICにもgoto～，gosub～returnの使用は許されています）。そのため今までお目に掛からなかった様な制御構造命令がいくつか存在しています。使いなれれば，そんなに面倒なものではありません。

if～then～else
for～next
while～endwhile
repeat～until
switch～case～default～endswitch

この中には皆さん良く知っているfor～nextやif～then～elseがありますが，これも前述の制御構造命令ですので良く覚えておいて下さい。詳細に付いては後で血の出るような特訓をするとして，例として，while～endwhileについてちょっと説明してみましょう。**第1－1図**の左側のプログラム（従来のBASIC）と右側のプログラム（X-BASIC）は全く同じ処理をするものです。

（従来のBASIC）

```
10    print i
20    i=i+1
30    goto  10
```

（X-BASIC）

```
10    int i
20    while 1
30    print i
40    i=i+1
50    endwhile
```

第1－1図　従来のBASICとX-BASICの違い

このように一見この制御構造命令を使用すると，2行多くなってしまうわけですが，内容を把握すれば単にそれだけでは無いことがわかります。と言うのは，行番号20だけを見ただけで，while～の方はこれ以降にあるendwhile迄の処理をループすることが判ってしまいますが，goto10の方は，行番号30を見るまではループになっていることが分かりません。

このことからわかる様に，制御構造命令をうまく使うことで，行番号で管理された世界から開放されるのです。また今までのBASICでは先にお話ししたサブルーチンの考え方が主流だったのですが，この考え方はそっくり関数（BASICプログラム内で関数定義が出来る）の考え方に代わってメインプログラムがすっきりして非常に分かりやすくなります。BASICの命令も関数化されていますので，プログラム全体を見てもはっきり意図がつかめるものになります。

1-2 X-BASICの特長

先にも述べましたが，X-BASICにはこれまでにない特長がいくつかあります。ここでは，その概要を説明します。

a）構造化されたプログラム形態

X-BASICでは，行番号に管理されないプログラムを構成するために制御構造命令が拡充されています。制御構造命令とは，プログラムの流れを支配するもので，従来はgoto，gosubが主流でした。しかし，goto，gosubは行番号を目標にしているため，先の行番号に管理されないプログラムを構成出来ません。X-BASICには行番号にとらわれること無く，プログラムの流れを変える制御構造命令が五つの用意されています。

●if～then～else
●for～next
●while～endwhile
●repeat～until
●switch～case～default～endswitch

上から二つ（if～then～else，for～next）は従来のBASICでも良く見かけます。これらの制御構造命令は，行番号を目標とする分岐をしないため，goto，gosubを使う必要がなくなります。

b）関　数

プログラムの流れは，前述の制御構造命令で管理されていますが，処理自身は，従来のBASIC同様処理に応じた内容のステートメント（命令）や演算等により行います。X-BASICでは，これまでのステートメントが，ステートメントと関数とに区別されていて，前述の制御構造命令以外のほとんどが関数化されています。具体的な使用方法としては以前のBASICと特別変わっている訳ではないのですが，関数であるため，その関数自身は値（戻り値）を持ち，その関数には何らかの引数があるということです。例えば，数学で学んだ三角関数などは，次のように記述します。ます。

$$X=COS(\theta)$$

ここでθが引数で，COS（θ）自身が全体で値になっています。これがXに代入される訳ですが，自分自身で持っている値を戻り値といいます。X-BASICでいう関数も，この考え方で今までのBASICで行っていたステートメント（命令）を使用してプログラムしていたものとほぼ同じ様に使用できます。
X-BASICに標準で用意（組込み）されている関数は，下記の通り4種類があります。これを，標準関数と言います。

●数値演算関数
●文字列処理関数
●データ変換関数
●ファイル入出力関数

これは，X-BASICに限らず，どのパーソナルコンピュータのBASICでもサポートされている基本的な処理を分類し，まとめたものです。しかし，X68000のハードウエアのスペックを見る限りこれだけの関数でユーザーは納得しません。みなさんが御存知の通り，X68000にはマシンの特長とも言える下記の様なハードウエア群があります。

●65536色のグラフィック
●最高256個まで使用出来るスプライト
●音声の録再生が出来るADPCM
●使い勝手の良いマウス
●ゲームにかかせないJOYSTICK
●ステレオ8重和音FM音源

これだけハードウエアが充実していれば，この機能をBASICで使わない手はありません。X-BASICでは，これらのハードウエアに対し，外部関数としてサポートしています。外部関数というのは，X-BASIC内には駐在してなく，X-BASICの起動時にBASIC.CNFというコンフィグレーションファイル（X-BASICの諸条件を設定するファイル）に登録されている外部関数（外部関数はハードウエア別に分類されている）を読み込み，X-BASICでこれらの外部関数の使用が出来る様になります。その他に，このBASIC.CNFでは，初期画面の一行の文字数の設定，X-BASICのプログラムや変数が格納されるメモリ（フリーエリア）の設定などができます。
X-BASICの関数は，これまでの標準関数や外部関数だけでなく，X-BASICプログラム中で既にある標準関数，外部関数，ステートメントや演算，代入等により関数を定義（func～endfunc）する定義関数，外部関数の構造と同様にマシン語（68000用のマシン語）により，作成した外部定義関数などがあります。

c）変　数

X-BASICは，基本的に四つの型の変数を扱うことが出来ます。変数というのは，ある数値に割

り当てられた名前のついた箱のようなもので，この箱の中に入る数値は，プログラムの進行とともに変化します。

```
      ∫
micom＝5               ……………… micomという変数に5を代入する。
      ∫
処    理
      ∫
micom＝micom＋3        ……………… micomという変数は5であったが
      ∫                           3を加えてmicomという変数へ再
                                  度代入され8となった。
```

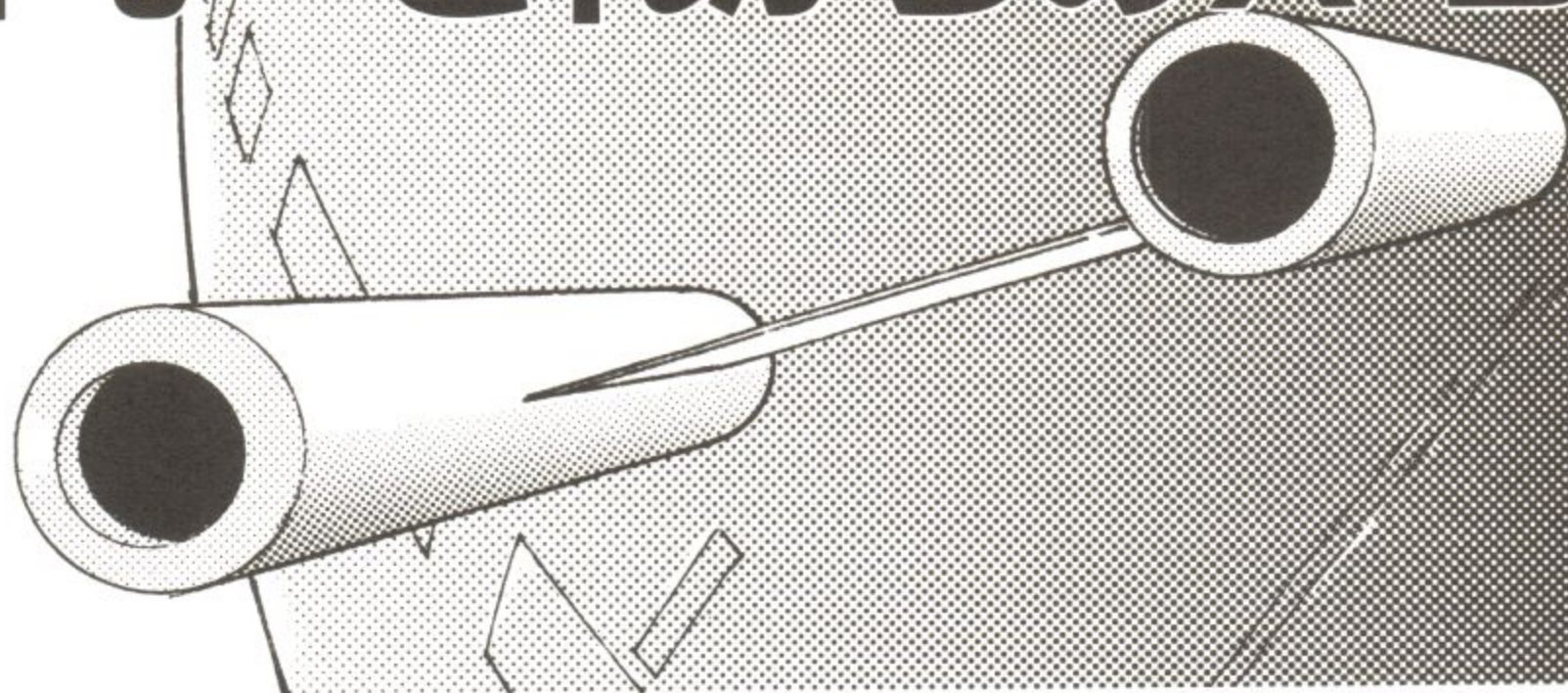
この様に，変数は数値と同様に扱う事ができ，数値を代入したり，演算過程に使用することができます。

さて，X-BASICでは，以下の四つの型の変数を使用することができます。

●int型 （4バイト，32ビット長）……………… −2,147,438,648〜2,147,483,648
●char型（1バイト，8ビット長）……………… 0〜255
●float型（8バイト，64ビット長）……………… $11,125,369,292,536\times10^{-308}$ 〜$35,953,862,697,246\times10^{308}$
●str型 （文字列）……………… 最大255文字まで

これらは，使用目的により，それぞれの型で変数を宣言し，プログラム中で使用します。

この変数には，別の分類があり，その変数の有効範囲により分けたものが，グローバル変数とローカル変数です。関数をプログラム内で定義した場合，その関数内で宣言された変数はローカル変数となり，メインプログラムからは，参照する事が出来ません。逆に，メインプログラム内で宣言された変数は，関数内でも参照できるグローバル変数となります。但し，メインプログラム内で宣言した変数名と同一のものを，関数内で宣言すると，関数内だけに有効なローカル変数となり，メインプログラムへ戻ったときには，先程のローカル変数は消滅しています。ローカル変数が使用可能になったため，再帰呼出しが出来る様になります。この他に，X-BASICで使用できる変数として，配列変数がありますが，これは少々ややこしいので，後で詳しく解説します。

1-3 これからのX-BASIC

X-BASIC自身の実行速度は，他の16bit機のBASICと同等のスピードです。従来のBASICでは，スピードを速くする手段として，BASICコンパイラを使用していましたが，コンパイル後のプログラムが非常に大きくなることと，全ての命令（ステートメント）を完全にサポートしていなかったり，コンパイル後の実行速度が思うように速くなかったりなどと，種々の問題が発生しています。これは，BASICの構造自身の問題で，もともとコンパイラを意識して作成されていない為，コンパイル時に無理が生ずるのです。

X-BASICは，制御構造命令を使用し，BASICのプログラム自身が構造化されている上，制作時に既にコンパイル方法も考えて完成されています。X-BASICは，BASICプログラムの形態で，行番号さえなければ，C言語とほとんどかわらないものがあります。従ってBASICのプログラム（ソースプログラム）は，Cソースコンバータで，Cのソースプログラムへ変換され，その後にCコンパイラにより，マシン語へコンパイルされます。従って，従来のBASICコンパイラで問題となっている「メモリ効率が悪い」，「コンパイル後の実行速度が，思うように上がらない」等の因子が，Cコンパイラに依存してきます。

Cコンパイラの能力は未知数であるとしても，今までのCコンパイラの概念や実績からいって，めちゃめちゃにメモリを食い，のろのろと遅いプログラムになる事はまずありません。このことから，X-BASICの将来は非常に明るいものになります。速度とメモリ効率は，このユニークな方法で克服されたのです。BASICの良い所としてプログラミング，即実行がありましたが，十分にデバックして（手直し）再実行をくりかえし，思う存分テストした上で，これで良いというところでコンパイルして実行速度を上げる非常に能率の良い方法になります。

具体的にX-BASICでのコンパイル手順を説明します。

[1] X-BASICプログラム作成

↓

[2] テストrun

↓

[3] X-BASICプログラムのデバッグ

↓

[4] basic to Cコンバータによりで言語のソースプログラムに変換

↓

5 Cコンパイラでアセンブルソースにコンパイル

↓

6 アセンブラ・リンカで実行可能なマシン語プログラムの完成

以上の過程で高速化したプログラムが完成しますが，1から3の部分は，X-BASICでの分担です。

ワンポイントテクニック　X-BASICのリストを大文字にする方法

X-BASICでは，リストを出すとき，forやprintなどの「予約語」や変数名が小文字で表示されますが，次のような作業を行うと，従来のBASIC同様，大文字表示にすることができます。

①いま利用しているX-BASICのディスクが，X68000を買ったときに付いてきたシステムディスクのマスターである場合は，まず，バックアップ（コピー）をとります。バックアップしたディスクには，ライトプロテクトシールを貼らないでおきます。

②いまビジュアルシェルが動作している時は，「COMMAND.X」をダブルクリックし，コマンドシェルを呼び出します。

次に，バックアップしたディスクをドライブAに入れ，

A＞ED BASIC￥BASIC.CNF↵

とします。A＞の部分はすでに表示されているので入力不要です。

③画面に数行の文字（下図参照）が表示されますので，この中の

画面左上に表示される内容

```
FREE    =128
WIDTH   =64
BEEP    =ON
CAPS    =OFF
FUNC    =AUDIO
FUNC    =GRAPH
FUNC    =MUSIC
FUNC    =MOUSE
FUNC    =SPRITE
FUNC    =STICK
```

CAPS　=　OFF

のOFFの部分をONに変えます。具体的に言うとカーソルを最初の(左の)Fの所まで持っていって，DEL DELと2回押し，Nと入力します。↵は不要です。

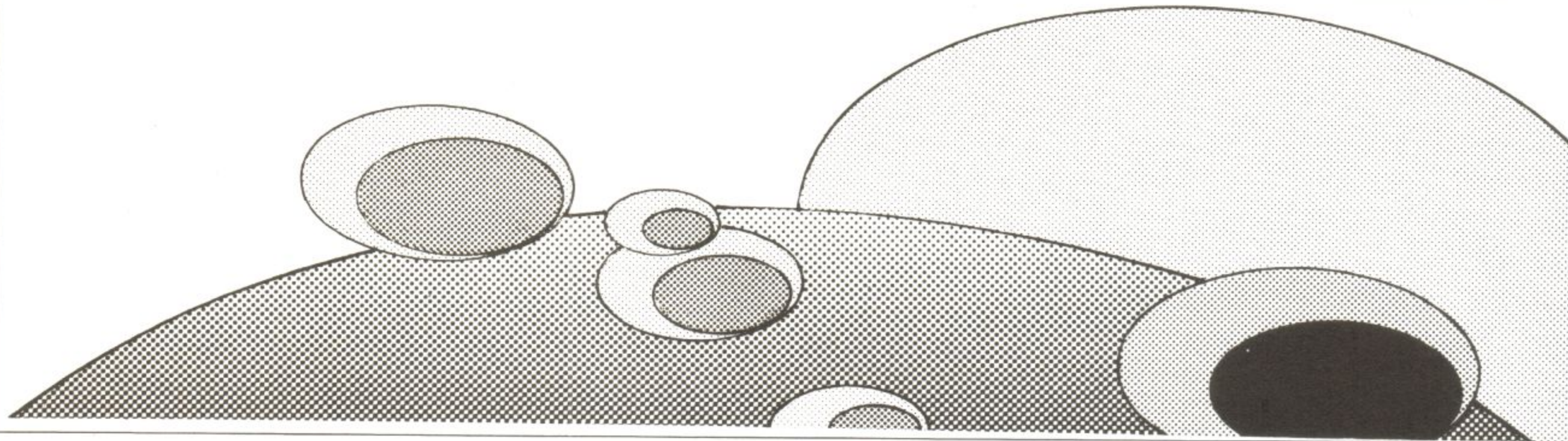
④注意深く，ESC Eと入力すると作業は終わりです。コマンドシェルに戻りますので，最初にビジュアルシェルを利用していた場合は

A＞EXIT↵で，ビジュアルシェルに戻れます。

いつもの手順でX-BASICを起動して，リストが大文字で表示されるか確かめてみて下さい。

第2章

X-BASICの基礎

2-1　X-BASICプログラミングの基礎

X-BASICの特長をいかしたプログラムを作成するためのポイントは,

- 変数
- 関数
- 制御構造命令

ですが,この他にもBASICの基礎概念として,エディットモード,プログラムの実行ファイル管理,等があります。これまでのBASICと方法が同じものもあれば,まったくちがった物もありますが,再確認するつもりで一通り説明します。BASICをある程度おわかりの方は,2－1は読み飛ばしていただいて結構です。

ここで,この後の説明を一定の基準で統一しておかなければならないので,使用するフロッピーディスクを一枚用意して下さい。このフロッピーは,システムディスクをそのままコピーしたものです。

2-1-1　ダイレクトモードとプログラムモード

X-BASICはHuman 68 kのビジュアルシェルからでも,コマンドシェルからでも,BASICを起動することが出来ます。起動された直後の画面は,**第2－1図**の様になります。

```
X-BASIC for X68000 version 1.00
Copyright 1987 SHARP/Hudson
  128Kbytes free

OK

█
```

第2－1図　X-BASIC起動画面

白い四角の点滅のことをカーソルといいます。このカーソルが点滅している状態を,入力待ち

状態といいます。この状態で**第２－２図**の通りタイプインして見てください。

```
X-BASIC for X68000 version 1.00
Copyright 1987 SHARP/Hudson
  128Kbytes free

OK
PRINT "X68000"█
```

第２－２図　PRINT 文を入力

この状態で⏎キーを押すと，**第２－３図の様になります。**

```
X-BASIC for X68000 version 1.00
Copyright 1987 SHARP/Hudson
  128Kbytes free

OK
print "X68000"
X68000


█
```

第２－３図　実行画面

ここに紹介した例は，print という命令文（画面にある決まった書式により，文や数値や変数の内容等を表示する）をダイレクトに実行したものです。このように，直接命令文を実行させることをダイレクトモードといいます。ダイレクトモードでの実行は，その命令文の結果を即座に知りたい時や，ためしにその命令を使用し，動作を確認したいときなどに使用します。

命令文に使用することが出来る命令は，基本的には，ステートメント，関数，演算式，代入式ですが，このままでは一つの命令文しか実行できません。複数の命令文をダイレクトモードで実行する場合は，命令文と命令文を：（コロン）で接続し，つないでいくことができます。**第２－４図**の例を実行してみてください。

```
print "X68000":print "X-BASIC

X68000
```

```
X-BASIC


█
```

第2－4図　：（コロン）で二つの句文をつなぐことができる

さきほどの，print ～という命令文が，二つ実行されたことがわかります。この様に：（コロン）により，いくつかの命令文を接続したものを，マルチステートメントといいます。**第2－5図**の例も実行してみてください。

```
for i=0 to 10:print i:next

0
1
2
 ⋮
9
10
```

第2－5図　マルチステートメント実行例

となりますが，ここでまた新しい命令文が登場してきます。

for i ＝ 0 to 10…… for 以下 next までを変換 i の値を 0 から 1 ずつ変化させてループする。

next……………………i の値に 1 を加えて for 以下を繰り返す。 i が10を越えたら次へ進む。

というそれぞれの命令文です。命令文の内容（ステートメント等）については，後述するのであまり重要ではないのですが，三つの命令文が：（コロン）により，全て実行されている点に注意してください。このように，ダイレクトモードで色々な命令文を組み合わせて，実行させることが出来ます。ダイレクトモードで実行させたあと，前述の例で for ～の所で，0 から10までの変化を10から20までに変更して，またダイレクトモードで実行したいのですがどうしたらよいでしょうか。

画面上の例文を見て，その通りキーボードから入力し for ～の所だけ 0 を10に，10を20に変更して入力してもいいのですが，非常に能率が悪いのでカーソルキー（[↑][↓][←][→]キー）の[↑]キーを押して for i ……の所へ合わせ，[→]キーで 0 のところまで進め，[INS]キーを押し，1 をキーインし，また[→]キーで10の 1 の所へカーソルを進め，[INS]キーを押して 2 をキーインし，[↵]キーを押します。するとどうでしょう。**第2－6図**のようになりました。

```
for i=10 to 20:print i:next

10
11
12
13
14
15
16
17
18
19
20

■
```

第2－6図 エディット操作でプログラムを修正

今の一連のキー操作をエディットといって，X－BASICのプログラム入力時や今の様なダイレクトモードの命令文入力時など文の変更の時に大変便利です。

エディットで使えると便利な方法をまとめると次の様になります。

1．打ち込んでいる行で，カーソル位置より手前の文字を他の文字に変更する。

〔例〕 PLINT "X 68000 " ■ ⇨ PRINT "X 68000 " ■

カーソルキーの←キーでLまで戻し，LをRに書き換えて，→キーでもとの位置へ戻る。

2．打ち込んでいる行で，カーソル位置より手前の文字を削除する。

〔例〕 PRRINT " 68000 " ■ ⇨ PRINT "X 68000 " ■

カーソルキーの←キーで後のRまでもどし，DELキーを押し，→キーでもとの位置へもどる。又は，カーソルキーの←キーで後のRまたはIまでもどし，BSキーを押し，→キーでもとの位置へもどる。

3．打ち込んでいる行で，カーソル位置より手前の文字と文字の間へ文字を挿入する。

〔例〕 PINT "X 68000 " ■ ⇨ PRINT "X 68000 " ■

カーソルキーの←キーでIまでもどしINSキーを押しそのキーに赤いランプがついたことを確認し，Rと入力し，INSキーを押し，赤いランプが消えたことを確認し，→キーでもとの位置へもどる。

4．以前実行したダイレクトモードの文の内容（まだ画面上に表示されているもの）を変更し再実行する。

〔例〕 print "X 68000 ": print "X-BASIC" ⇨ print "X-BASIC": print"X-BASIC

X 68000　　　　X-BASIC

X-BASIC　　　　X-BASIC

■　　　　■

カーソルキーの↑キーで，変更したい行へもどし，1～3の要領で変更した後，その行で↵キーを押す。

さて，本題へ戻りますが，ダイレクトモードでの実行は，必ず↵キーを押すことにより実行を開始します。そして，実行後は画面にダイレクトモードの命令文が残っている場合を除き，再実行させる時にはもう一度同じ文をキーボードから入力しなければなりません。これは前にも述べましたが，一時的に動作を確認するためだったり，その命令文の結果を，即座に知るためなので，しかたがありません。では，どうして同じ命令文を何回も実行させたら良いのでしょう。さきほどの

print "X 68000 ": print "X-BASIC"

の文を使用して説明します。エディットの機能を利用して頭文字の前に10という数字と␣(スペース）を一つ打ち込んでください。

10　print "X 68000 ": print "X-BASIC"

となります。ここでダイレクトモードのときのように↵キーを押しますが，画面上に

X 68000

X-BASIC

■

というようには出力されませんでした。しかし，ここで**第2－7図**の様に入力し，↵キーを押すとどうでしょう。

```
10  print "X68000":print "X-BASIC"
RUN

X68000
X-BASIC

OK
■
```

第2－7図　行番号のついたプログラムを実行する

ダイレクトモードで実行した時の様に，画面に文字が出力され，OKとなりカーソルが現れま

す。この後RUN⏎キーを押せば，何回でも”X 68000 ”と”X-BASIC”が画面に出力されます。これは，ダイレクトモードの命令文に，行番号をつけたために，プログラムモードで

```
10　print ”X 68000 ”　: print ”X-BASIC”
```

という命令文が，X-BASICのプログラムメモリに書き込まれたためです。即ち，行番号␣命令文（：命令文：命令文…）の形式になると、プログラムモードと解釈され，プログラムメモリへ格納されます。プログラムメモリ内に入っているプログラムに対する操作は，全てX-BASICのコマンドにより行います。例えば，今のRUNというのは，コマンドで”プログラムメモリに書き込まれているプログラムを実行せよ”という意味になります。コマンドには，次の様なものがあります。

コマンド一覧

RUN	プログラムメモリ内のプログラムの実行。
LOAD	指定されたファイルネームで，ディスク内に記憶されているプログラムをプログラムメモリに移す。
SAVE	指定されたファイルネームで，プログラムメモリからディスクへ記憶する。
NEW	プログラムメモリ内を，クリアする。
RENUM	指定された現在の行番号から，指定行番号間隔で，指定行番号から行番号をふり直す。
DELETE	指定された行番号から，指定された行番号までを，プログラムメモリから消去する。
LIST	プログラムメモリ内のプログラムを，画面に出力する。
FILES	ディスク内のファイルを，画面に出力する。
CONT	BRAKE で停止したプログラムを続行する。
CLEAR	宣言されている変数をすべて解除する。
KEYLIST	ファンクションキーのリストを画面に出力する。
CHILD	チャイルドプロセスへ入る。

WIDTH	画面の1行文字数と行数の指定。
SYSTEM	親プロセスへもどる。

コマンド一つ一つの動作は後述しますが，基本的にこれらのコマンドはプログラム内へ使用することが出来ません。常にダイレクトモードでの使用ということになります。その他の制限としてダイレクトモードでこのコマンドをマルチステートメントで記述してはいけないということです。

ダイレクトモードでの命令文に行番号をつけたものが，プログラムとしてプログラムメモリに記憶されることがわかったわけですが，行番号の役割とは，いったい何でしょうか。第1章でも説明しましたが，X-BASICでは「構造化されて，行番号に管理させないプログラムを作れる。」としましたが，実際には行番号は，プログラムメモリ内の住所の役割をしています。例えば，**第2－8図**を参照してください。

```
10 int i
20 for i=0 to 10
30 print i
40 next
50 end
```

第2－8図　10～50までのプログラム

このリストは，先程ダイレクトモードのマルチステートメントで1行で実行させていたものを展開して，プログラムモードにしたものですが，30 print iの行のことを，「上から3行目の文」というのと「30行目の文」というのでは，後者の方がはっきりと目的とする行を言うことが出来ます。

また，このリストは，50行目まで10番とばして5行しかないので，「上から3行目」でも良いのですが，何千行ものリストで「上から352行目」なんて言われても，上から順に数えていかなければなりません。また，コマンドを思い出してください。

DELETEについても，「上から何行目から何行目までを削除」なんて言っても，本当にその行からでいいか確認するには，上からかぞえなければなりません。行番号は，その文の住所を知るための住所番地なのです。その他の役割としては，プログラムを進行させる順序を知らせる役割もあります。**第2－9図**の例のように，打ち込んでみて下さい。

```
NEW ↵
10 int i
30 print i
```

```
50 end
40 next
20 for i=0 to 10

█
```

第2－9図　行番号をめちゃくちゃに入力する

ここでRUN ⏎と打ち込むと0から10まで出力してカーソルが点滅して、入力待ちの状態となります。この時LIST ⏎と打ち込むとどうでしょう。

```
LIST ⏎
10 int i
20 for i=0 to 10
30 print i
40 next
50 end
OK
█
```

第2－10図　きちんと順番になっている

第2－10図のようにプログラムが出力されます。先に入れた順序でなくて，行番号の小さい順に並びかえて出力されています。これは，プログラムは行番号の小さい順に処理を進めていくからで，キーインによりプログラムメモリへ，入力していく順序ではないからです。したがって，プログラムの変更時にある行と，その次の行の間に別の処理を入れたい場合は，画面上（なるべく左端）のなにも書いてない行へ，行番号と行番号の間の整数を行番号とした命令文を入力すればいいのです。たとえば，上の例で，30と40の間にもう一つ print ～を入れる場合は，

```
10 int i
20 for i=0 to 10
30 print i
40 next

50 end
```

```
35 print "X68000"          ・・・・・・      新しく追加した行
■
```

第２－11図　35行を後から入力する

第２－11図のように入力して LIST↵をキーインすると

```
LIST
10 int i
20 for i=0 to 10
30 print i
35 print "X68000"
40 next
50 end
```

第２－12図　30行と40行の間に35行が追加される

第２－12図のようになります。これを行の挿入といいますが，このような場面を常に予測してプログラムを入力する際は，必ず10行単位や100行単位で行番号を割り付ける様にしています。

行番号␣命令文（：命令文：命令文…）の形式によりプログラムと解釈されてプログラムメモリに格納され，割り当ててない行番号にプログラムを書くと，行番号の前後関係により定まる位置にプログラムが挿入されることがわかりました。次はプログラムの削除です。先程の画面をそのまま利用します。

第２－12図の画面より，35↵と打ち込み，LIST↵とします。

```
35
LIST
10 int i
20 for i=0 to 10
30 print i
40 next
50 end

■
```

第２－13図　35行が削除された

今度は，先程の35行の文がなくなっていることがわかります。このように，行全体を削除する

場合は，削除したい行番号⏎を実行します。この他に，X-BASICのコマンドのDELETEを使用した行の削除がありますが，このコマンドは，連続した複数行の場合に有効です。

2-1-2 コマンド

さて，このぐらい説明が進んでくると，X-BASICのプログラミングでエディット時にダイレクトモードで使用するコマンドを使わなければ，いろいろと説明しにくくなりますので，コマンドの説明をします。この説明の中では，今後進めていく上での，コマンドのキーワードも説明しますので，注意してください。

形式	RUN	基本形
	RUN　行番号	行番号指定
	RUN　"ファイルネーム"	LOAD＋RUN

RUNというのは基本的にはプログラムメモリ内のプログラムを実行させるコマンドです。プログラムは，行番号の小さい順に実行されるため，基本形ではプログラムメモリのなかで一番小さい行番号から実行します。プログラムの途中から実行するには，RUN 行番号という形式の行番号指定の実行を行います。この他に，ファイルネーム指定の実行がありますが，これはLOADのあとにRUNを実行したものと同じですので，LOADの項を参照して下さい。本文中"このプログラムを実行すると……"とありましたら，このRUNの事です。

形式　LOAD　"ファイルネーム"
　　　LOAD @　"ファイルネーム"，開始行番号，増分

現在，ディスク等に登録されているX-BASICのプログラムを，プログラムメモリへ転送します。X-BASICは，プログラムメモリ内のBASICプログラムのみ実行可能ですので，ディスク等に格納されているプログラムを一度プログラムメモリへ転送する必要があります。その後，そのプログラムを修正するなり，実行するなりの手順となるわけです。LOAD @ "ファイルネーム"については，プログラムの結合に使用されるので，後述します。このコマンドを実行すると，今までプログラムメモリに入っていたプログラムは消去されます。本文中に，"プログラムをロードし……"とある場合は，このLOADコマンドで特に指定しない限りファイル名は任意のファイル名で結構です。

形式　SAVE　"ファイルネーム
　　　SAVE @　"ファイルネーム"，開始行，終了行

現在，プログラムメモリ内にあるX-BASICのプログラムを，ディスク等へ転送し保存します。しかし，ディスク内に登録する際に，OSが管理する対象となるファイルネームが必要となるので，

必ずファイルネームを指定して下さい。SAVE @ ”ファイルネーム”は，プログラムの結合に使用されるので後述します。本文中に”プログラムネームをセーブし……”とある場合は，このSAVEコマンドでファイル名についてはLOADと同様です。

```
形式　NEW
```

これは，プログラムメモリ内にあるX-BASICのプログラムを消去するコマンドです。同時に，宣言されていた変数はすべて解除されます。新しいプログラムを入力する時には，必ずNEWするように心掛けて下さい。

```
形式　RENUM
　　　RENUM　新行番号
　　　RENUM　，旧行番号
　　　RENUM　，，増分
　　　RENUM　新行番号，，増分
　　　RENUM　新行番号，旧行番号
　　　RENUM　，旧行番号，増分
```

RENUMとはRenumberのことで，プログラムに付けてある行番号の操作をします。具体的には，行と行の間に行を挿入して間隔をつめたり，プログラムの一部行番号をつけかえて，プログラム内の位置をかえたりします。指定することが出来るのは，新しくつける行番号の先頭と現在の行番号のどの位置からかと，新しくつける行番号の増分ですが，指定ナシということも出来ます。新行番号を省略すると10からになり，旧行番号を省略すると現在のプログラムの先頭からとなり，増分を省略すると増分が10となります。

```
形式　DELETE　開始行ー終了行
　　　DELETE　開始行ー
　　　DELETE　ー終了行
```

プログラム内の指定行番号から，指定終了行番号までを削除します。だだし，終了行番号を省略した場合は終わりまでとし，開始行番号を省略した場合は最初からとします。削除されるのは，開始行と終了行も含まれるので間違いのないようにして下さい。本文中に，”何行目から何行目までを削除して……”とある場合は，このコマンドを使用します。（※注）（行番号だけ打ち込んで，一行ずつ削除する方法でも構いません）

形式 LIST
　　LIST 開始行ー
　　LIST ー終了行
　　LIST 開始行ー終了行

プログラムメモリ内のプログラムを画面へ出力します。範囲の指定がない場合にはプログラムすべてが，また開始行や終了行のどちらとも指定した場合は指定された範囲が画面に出力されます。LISTは頭にLをつけ，プリンタにプログラムを打ち出すこともできます（LLIST）。本文中で，”リストを見て下さい……”とある場合は，打ち込んであるリストかまた本文で紹介しているリストかに注意して下さい。

形式 FILES
　　FILES ”デバイス名：ファイル名”

FILESというのはデバイス（一般にはディスク）内のファイルの一覧を画面に出力するコマンドです。ここで，FILESの後に”（ダブルクォート）でくくったデバイス名，ファイル名を指定するとある限られた範囲のファイルを検索し表示します。限られた範囲というのは，例えば”BドライブのBINというディレクトリ内”とか，”Aドライブのルートディレクトリ”というように指定されたということです。また，ファイル名の指定にはワイルドカードを使用でき，次のような記述になります。ワイルドカードについては，本体のHuman 68 Kユーザーズマニュアルを参照して下さい。

```
    files    "b:*.*"
デバイス名 ─┘ └ワイルドカード
```

色々な応用例がありますが，一般的には次のような記述を覚えていれば十分です。

```
    files ↵
    files "a:"
           └──ドライブ名
      files "a:¥BIN"
ドライブ名┘   └ディレクトリ名
```

形式 CONT

X-BASICのプログラムを実行中に，BREAKキー，CTRL+Cキー，およびプログラム中のSTOPステートメントで実行を中断したときに，CONT↵とすることで再開することができます。これら三つの方法で中断したときに，ディスプレイにはどこで中断したかが表示されます。ただし，中断しているときは入力待ち状態でカーソルが点滅しているので，リストやprint文により変数の表示，確認はできますが，プログラムの変更をすると再開できなくなります。

形式　CLEAR

現在，宣言されている変数を初期化（X－BASIC起動時の状態）します。例えば，次のようにダイレクトモードでｉという変数を整数32ビットという型に宣言します。

int i ↵

■

現在，ｉは整数32ビット型の整数として宣言されています。ここで，もう一度次のように打ちこみます。

int i　　　　一回目の宣言

int i ■　　　　今，入力しようとしている命令文

ここで，↵キーを押すとエラーを知らせるビープ音（”ピッ”という音）と共に，画面に「変数はすでに宣言されています」と表示されます。これは言うまでもなく，一回目の宣言でｉという変数は整数32ビット（int型）に宣言されているためで，もう一度宣言しようとしたためにX－BASICが注意をうながしたのです。では，ここで次のようにします。

int i

int i

変数はすでに宣言されています

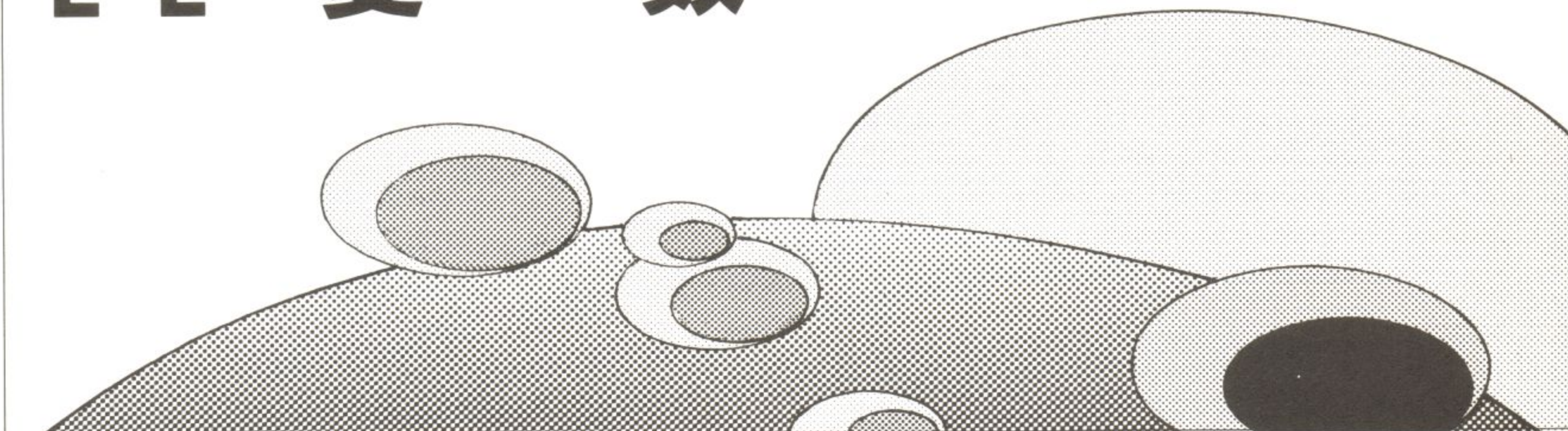
clear ↵

■

とすることで，ｉという変数の宣言は解除されたので，もう一度”int i↵”としても今度はエラーにはならないのです。

形式　KEY　LIST

このコマンドはファンクションキーの内容を画面に出力するときに使用します。ファンクションキーとは，キーボードの上部に位置する10個のキーで，キートップに［F1］～［F10］と表示されているものです。このファンクションキーは，一つのキー操作で［F1］～［F10］それぞれに登録されている文字列をキー入力したのと同じ結果が得られる，とても重宝なものです。［F1］～［F10］のキーは，［SHIFT］キーを併用することで［F11］～［F20］に対応していて，計20個のファンクションキーがあります。

2-2　変　　数

2-2-1　変数の型と宣言

X-BASIC には，四つの型の変数があります。それぞれの型はプログラム中で変数の用途により使いわけをします。

a）char 型変数

char 型変数は，0 ～255までの整数を扱うことができます。主に文字コードを扱うような場合に使用されます。文字コードと言うのは，文字をコンピュータが扱いやすくするために文字に数値を１対１対応で割り付けたもので，一般的に char 型の変数１個で半角文字（８×８ドットや８×16ドットの文字）の英数，カナ記号が表現できます。日本語の場合は漢字という恐ろしい文字があるので，char 型の変数一つでは表現しきれずに，二つ使用して１文字の全角文字を表現します。

さて char 型の変数として，アルファベットの a を宣言してみます。

```
char a
```

これで，X-BASIC で a という変数は char 型に宣言されました。この宣言の方法はダイレクトモードでの宣言ですが，プログラムモードでは

```
行番号␣char a
```

という形式になります。基本的には，これ以後プログラム中では a という変数を再び宣言することはありません（例外として，定義関数内で，再定義するとローカル変数として扱われ，関数をぬけた所で，関数内の変数は削減します)。

b）int 型変数

int 型変数は－２,147,483,648～２,147,483,648までの整数を，扱うことができる変数の型です。この型の変数は，X－BASIC のプログラム中では，一番使用頻度が高く，使用目的も多様です。整数であるという制限はありますが，±21億の数値ですから，ほとんど桁があふれる（オーバーフロー）の心配はありません。

この変数の宣言は，次の様にします（ダイレクトモード）。

```
int x
```

これを，プログラムモードで宣言するには，

```
行番号␣int x
```

とすれば良く，char型の変数同様，これ以後この変数の宣言は，基本的にはする必要もなく，またしてはいけません。

c） float型変数

float型の変数は，約1.1×10^{-308}～3.6×10^{308}の実数の扱いをゆるされています。これは，科学技術計算等の計算に，精度が必要な場合に使用されます。

この変数の宣言は，次の様にします（ダイレクトモード）。

```
float y
```

これもプログラムモードで宣言するには，先頭行番号を付けて表記します。

d） str型

str型の変数は，char，int，floatの型とは異なり，文字列を扱うことのできる変数です。文字列というのは，文字を表すコード（char型の時に説明したもの）が並んで，単語になった状態とか，その1文字でもかまいませんが，最後にその文字列の終わりを表す0（ヌルコード）を付けたものをいいます。

例えば，X-BASICという文字列の内容を解剖してみると**第2－14図**のようになります。

	X	-	B	A	S	I	C	
	↓		↓	↓	↓	↓	↓	
16進数	&H58	&H2D	&H42	&H41	&H53	&H49	&H43	&H00
10進数	88	45	66	65	83	73	67	0

文字列最後を表す0(ヌルコード)

第2－14図　文字列の解剖

さて，str型の変数には，このようにいろいろな文字列を格納することができますが，文字列の長さには制限があります。最大で255文字まで格納することができます。さて，このstr型の変数の宣言ですが，次のようになります。

```
str m [10]
```

これまでのchar，int，floatとは違って，後ろに［10］というものがついています。これは，

いま宣言したmというstr型の変数は，最大で10文字までということです。従って，プログラム内で扱う文字列が，50文字以下となれば，つぎのようになります。

str j [50]

ここで，[50] という文字数の宣言をなくし，次のように表現します。

```
str n
```

これは，今までの3種の数値型の変数と，同じ形式になりましたが，このままでも，32文字までの文字列がゆるされます。従って，文字数の宣言を省略すると，

str p [32]

と同様になります。プログラムモードでは，行番号を付けて表記することは，いうまでもありません。

2-2-2 変数の初期化

プログラム内で，いろいろな変数を使用して，プログラムを作成していく場合，それぞれの変数に，値を代入しなければなりません。ここで，**第2－15図**のプログラムを見て下さい。

```
10 int x ·········xという変数を int 型に宣言
20 int y ·········yという変数を int 型に宣言
30 x=10: y=15····xに10を代入、yに15を代入（マルチステートメント）
40 print x········xの内容をディスプレに出力
50 print y·······yの内容をディスプレに出力
60 print x+y·····xの内容とyの内容を加えてディスプレに出力
70 end············プログラムの終了
```

第2－15図　変数の初期化

ここで，xとyという変数二つが，登場してきます。

```
10 int x=10,y=15
20 print x
30 print y
40 print x+y
50 end
```

第2－16図　初期化

これを**第2－16図**のように，行番号10で二つの変数と宣言と，初期化を同時に行っています。このように，同時に，同型一つの型の宣言を意味するキーワード（int，float，char，str）に続いて，複雑の変数を，(コンマ) で区切り，つづけて表記することができ，同時に＝（イコール）数値又は文字列の形で，初期化をすることができます。

さて，このプログラムの実行結果はみなさんが想像しての通りです。

```
10 int x=10,y=15
20 print x
30 print y
40 print x+y
50 end
RUN
10
15
25
OK
█
```

第2－17図　プログラム実行

第2－17図のようになるわけです。

それでは次に，str 型の変数の初期化はどうすればいいのでしょうか。これまでは数値型の変数だったので，非常にわかりやすかったのですが，文字列となるとちょっと分かりにくくなります。X-BASIC で文字列は次のように表されます。

"X-BASIC"

実際の中味となる部分は，"（引用符）を取除いた X-BASIC になりますが，文字列であることを意味するために"（引用符）でかこみます。

それでは，str 型の変数に文字列を代入してみます（**第2－18図**）。

```
10 str a="X-BASIC",B="X68000"
20 print a
30 print b
40 end
```

第2－18図　str 型の変数の宣言

行番号10で a と b の変数を文字列用の変数として宣言し，なおかつ文字数の指定がないので最大32文字までとなり，同時に a には "X-BASIC" が，b には "X 68000 " が代入されまてした。しかし，ここで注意しなければならないのは，str 型変数のところで説明したように，文字列の終わりには自動的に 0 （ヌルコード）が入っているという事です。

このプログラムの実行結果は，**第2－19図**の通りです。

```
10 str a="X-BASIC",b="X68000"
20 print a
30 print b
40 end
RUN
X-BASIC
X68000

OK
■
```

第2－19図　プログラム実行

さて，文字列"X-BASIC"がaという変数で"X 68000"がbという変数に代入され，aとbで表現されていると，あたかも数値であるような錯覚におちいりますが，文字列の場合は加算（結合）だけは許されています。今のプログラムに，**第2－20図**のように35行を追加してみて下さい。

```
10 str a="X-BASIC",b＝" X68000"
20 print a
30 print b
35 print a+b
40 end
```

第2－20図　第2－18図に35行を追加する

これを実行すると，

```
RUN
X-BASIC
X68000
X-BASIC X-68000

OK
■
```

第2－21図　文字列の結合

第２－21図のようにａとｂとの内容が，結合して出力されることがわかります。ｂ＋ａとすれば，

X 68000 X-BASIC

ということになります。

2-2-3 配列変数

ａ）一次元配列

話が少々具体的になりますが，生徒の数が30人のクラスがあったとします。30人が，国語のテストをして，先生が採点し，集計にＸ－BASICでプログラムを作り計算させようと思いました。それでは，それぞれの生徒の変数を割り当ててみます。

```
10 int a ＝30,  b ＝25,  c ＝50, ……
```

あれ？生徒は30人ですが，アルファベットは26個しかありません。４人分の変数がたりません。X-BASICでは，複数のアルファベットの並びを変数とすることも許されていますが，ここでは１文字のアルファベットしか使えないものとします。これでは，30人分のデータを変数に格納して集計させることはできません。仮に，生徒が25人で，Ａ～Ｙまでの変数で足りたとします。しかし，Ｋという変数は，いったい誰なのでしょう。

普通，生徒には出席番号なるものがありますので，出席番号と生徒たちは，１対１に対応しています。このような場合に，配列変数を使用します。

配列変数は，変数の後に（　）（カッコ）で囲んだ数字を入れて表現します。
例えば，

X (10)

（　）の中の10は，別に数字とは限らず，次のように変数の場合もあります。

X (I)

この配列変数を宣言するには，変数宣言の先頭に，dim（ディメンション）とつけ加えればいいのです。

```
10 dim int X (20)
```

このとき（　）カッコの中の数値は配列の最大値で，この場合は０から20まで21個の変数が宣言されたことになります。これを，先程の国語のテストの点，名前，出席番号として，表にまとめてみると，**第２－22図**のようになります。

氏名	出席番号	(変数)	点数	
****	0	X(0)	****	一般的に出席番号には0はない。
秋山	1	X(1)	50	
伊藤	2	X(2)	35	
上野	3	X(3)	82	
.	.	.	.	
.	.	.	.	
.	.	.	.	
片山	18	X(18)	65	
.	.	.	.	
.	.	.	.	
.	.	.	.	
深沢	28	X(28)	72	
町田	29	X(29)	22	
森	30	X(30)	19	

第2－22図　配列変数と名前，出席番号，テストの点数の関係

この時先生が，

10 dim int X (30)

と宣言すれば，めでたく出席番号で変数を指定でき，そこへ国語のテストの点を代入していくことができるのです。

配列変数の初期化は，次のようになります。

10 dim int X (30) ＝ {0，50，35，82，……，65，……，72，22，19}

中カッコ { } の中に，配列変数のそれぞれの要素に対応する初期値を，(コンマ) で区切っていけば,それぞれの配列変数が初期化されます。上の例を,ていねいに記述してみます(**第2－23図**)。

```
10 dim int X(30)
20 X(0)=0
30 X(1)=50
40 X(2)=35
50 X(3)=82
 ⋮   ⋮   ⋮
200 X(18)=65
 ⋮   ⋮   ⋮
400 X(28)=72
410 X(29)=22
```

```
420 X(30)=19
```

第2－23図　配列変数の初期化

b）2次元配列

今説明した配列は，国語だけの1教科でしたが，このクラスで実施したテストが中間テストなどで，他に数学，理科，社会，英語などもテストしたとするとどうでしょう。

X (30) が国語用の配列変数なので，Y (30) を数学とし，Z (30) を理科，V (30) を社会，W (30) を英語として，次のように配列変数を宣言します。

　10 dim int X (30)，Y (30)，Z (30)，V (30)，W (30)

しかし，この方法で教科がもっと増えてきたりすれば，変数の割り当てや，その変数がどの教科なのか，わかりにくくなってしまいます。そこで登場するのが，2次元配列です。2次元配列は，1次元配列が二つあると考えればよく次のような宣言します。

　10 dim int X (5,30)

最初の5は，0～5までの数値が変化できるということで，教科を表します。その後の30は，生徒の数です。従って，次のように教科と生徒の出席番号をまとめることができるのです（第2－24図）。

		国語	数学	理科	社会	英語
	出番／教科番号	1	2	3	4	5
秋山	1	X(1,1)	X(2,1)	X(3,1)	X(4,1)	X(5,1)
伊藤	2	X(1,2)	X(2,2)	X(3,2)	X(4,2)	X(5,2)
上野	3	X(1,3)	X(2,3)	X(3,3)	X(4,3)	X(5,3)
¦	¦	¦	¦	¦	¦	¦
片山	18	X(1,18)	X(2,18)	X(3,18)	X(4,18)	X(5,18)
¦	¦	¦	¦	¦	¦	¦
深沢	28	X(1,28)	X(2,28)	X(3,28)	X(4,28)	X(5,28)
町田	29	X(1,29)	X(2,29)	X(3,29)	X(4,29)	X(5,29)
森	30	X(1,30)	X(2,30)	X(3,30)	X(4,30)	X(5,30)

※出番＝出席番号　教番＝教科番号

第2－24図　5教科のテストを2次元配列で表す

この2次元配列の初期化も1次元配列同様に｛｝の中に，その初期値を，(コンマ）で区切って設定します。

```
10 dim int X(5,30)={0,0,0,····,0,0,0            X(0,?)の部分
20                  0,34,25,····,85,32,18
30                  0,48,25,····,62,60,38
¦                               ¦
50                   ·····················}
```

第2－25図　2次元配列の初期化

ここで，注意しなければならないのは，配列を宣言する時に，dim int X（1）とすれば，X（0），X（1）が存在することです。初期化のときに，X（0）を忘れると，全部1列ずつずれてしまうということです。

c）高次元配列

1次元，2次元ときて，つづいてその他の配列ですが，宣言の仕方は今までと特に代わりません。

```
dim int X（8，5，30）
```

先程の例に，8クラスのテストの集計ができるようにしたと思えばいいのです。いいかえれば，

```
X（5，30）
```

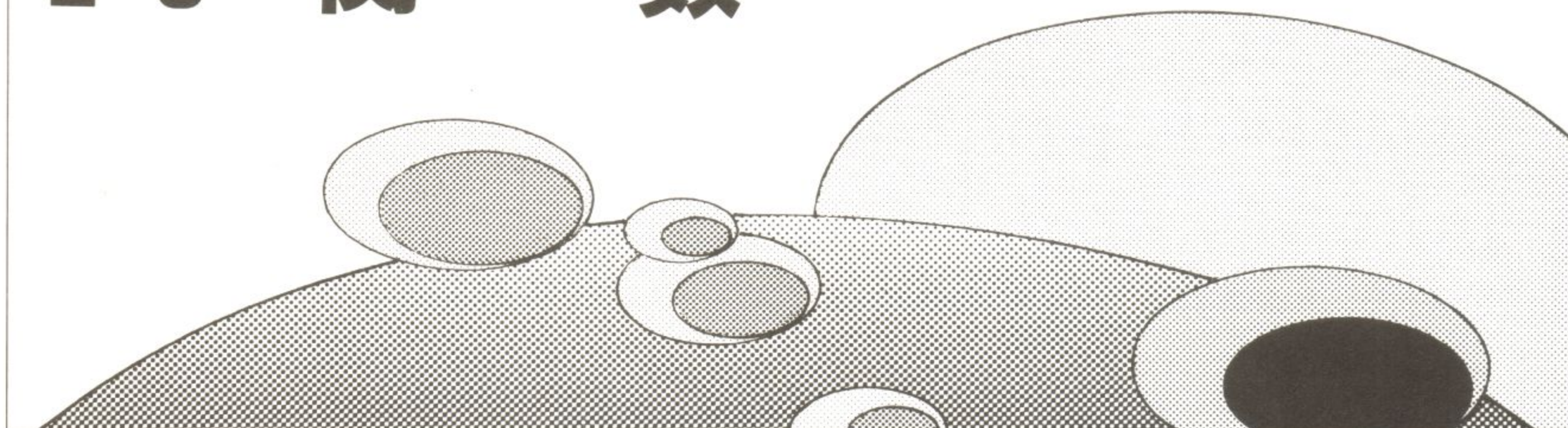
という配列が8個あると思えばいいのです。この配列をことばで表現すれば，「1クラス30人で，8クラスで実施した5教科のテストの点数を格納することができる変数」となります。X-BASICの文法的な制約としては，高次元配列は255次元まで扱うことが可能なのですが，X-BASICの1行の文字数の制限が255文字なので，そのような長い配列を宣言する文が書けないのはわかっていただけると思います。

余談となりますが，8クラスで1学年ならば，1年生から3年生までも格納できる配列を，4次元配列で可能にできます。

```
dim int X（ 3，   8，    5，    30  ）
           ↓     ↓      ↓      ↓
          学年  クラス  教科  出席番号
```

2-2-4 その他の変数の注意

変数は，四つの型の他に，ローカル変数とグローバル変数というわけ方があります。これは，前述の四つの型とは関係なく，プログラム中にその変数がどの位置にあるか，またどの位置で宣言されたかにより決定されます。これは，従来のBASICでは変数は使いすてのものだったのですが，X-BASICではプログラム中で使用する変数は必ず宣言しなければならないので，プログラム作成時にはプログラマーの管理のもとで，それらの変数を扱わなければならないということです。具体的にいうと，メインプログラム内と関数内で宣言された変数のことで，それぞれはまったくの異次元で，その橋渡しは引き数と戻り値です。詳しくは，2－3－4で解説します。

2-3　関　　数

2-3-1　関数の考え方

数学でいう関数とは，例えば

y＝f（x）

と表現します。「f（x）＝ax＋bのグラフを書きなさい」というような時は，一般的には最初にx＝0を代入し，グラフ上にf（0）を計算してプロットし，次にx＝1を代入し，f（1）の計算によりもう一つグラフ上にプロットしてそれを直線で結びました。これは，y＝ax＋bというのが，直線の方程式であることを知っているためで，実際はf（2）と表現しているので，その内容はわかりません。そんな理由かどうか，関数のことをある数値を入れると，何か加工をして出力してくるブラックボックスであると言われることもあります。

ここで重要なことは，「あるものを入れると，何か加工されて出力してくる」という単純なことです。この「ある物を入れる」という入れるものを**引数**といい，「加工されて出力してくる」の出力は**戻り値**といいます。

引数は，f（x）のxのことで，f（x）全体が戻り値となります。y＝f（x）と表現したy＝のyはあくまでも戻り値をyに代入したもので，f（x）全体で戻り値となるのです。

さてX-BASICでは，様々な命令（X68000の諸機能を使用する）を関数の形で実現しています。また，X-BASICでは与えられた関数（付属のもの）以外にも，自分で定義した関数を使用することができます。

2-3-2　関数の種類

X-BASICの関数の種類は，大きくわけて次の4種類があります。

1）標準関数

これは，X-BASIC内に初めから内蔵されていて，文法さえ間違えなければ何の定義や宣言等をしなくても使用できる関数です。関数の内容は，主にX-BASICの基本的な操作，演算，ファイル操作，データ変換などです。

2）外部関数

これは，X 68000のもつ機能を充分発揮できるように，X 68000のハードウエアに，密着した内容の関数です。この外部関数は，X-BASICの起動時に，”BASIC. CNF” というファイルに，登録されている外部関数を組み込むため，初めからX-BASICのプログラム中には入っていません。外部関数は，第２－26図のような種類に分類されていて，X 68000の機能別に管理されています。

		ファイル名
・グラフィック関数		GRAPH. FNC
・スプライト関数	（スプライト）	SPRITE. FNC
・ミュージック関数	（ＦＭ音源）	MUSIC. FNC
・オーディオ関数	（ＡＤＰＣＭ）	AUDIO. FNC
・マウス関数		MOUSE. FNC
・ジョイテック関数		STICK. FNC

第２－26図　X 68000の外部関数

従って，”BASIC. CNF” に登録されていない関数は使用できません。これらの関数群は，～. FNCというファイルでシステムフロッピーのBASICのディレクトリ内にあります。

３）定義関数

標準関数や外部関数や，その他のステートメントにより，プログラムを作成する際，従来のBASICのサブルーチン的なものを，X-BASICプログラム中に関数として定義することができます。これは，ステートメントのfunc ～ endfuncを使用することにより，関数化することができます。

４）外部定義関数

外部関数は，68000用のアセンブラにより作成されていますが，ユーザーの努力しだいで外部関数同様に，自分だけの外部関数を作成することができます。この方法については，第４章で詳細を説明します。

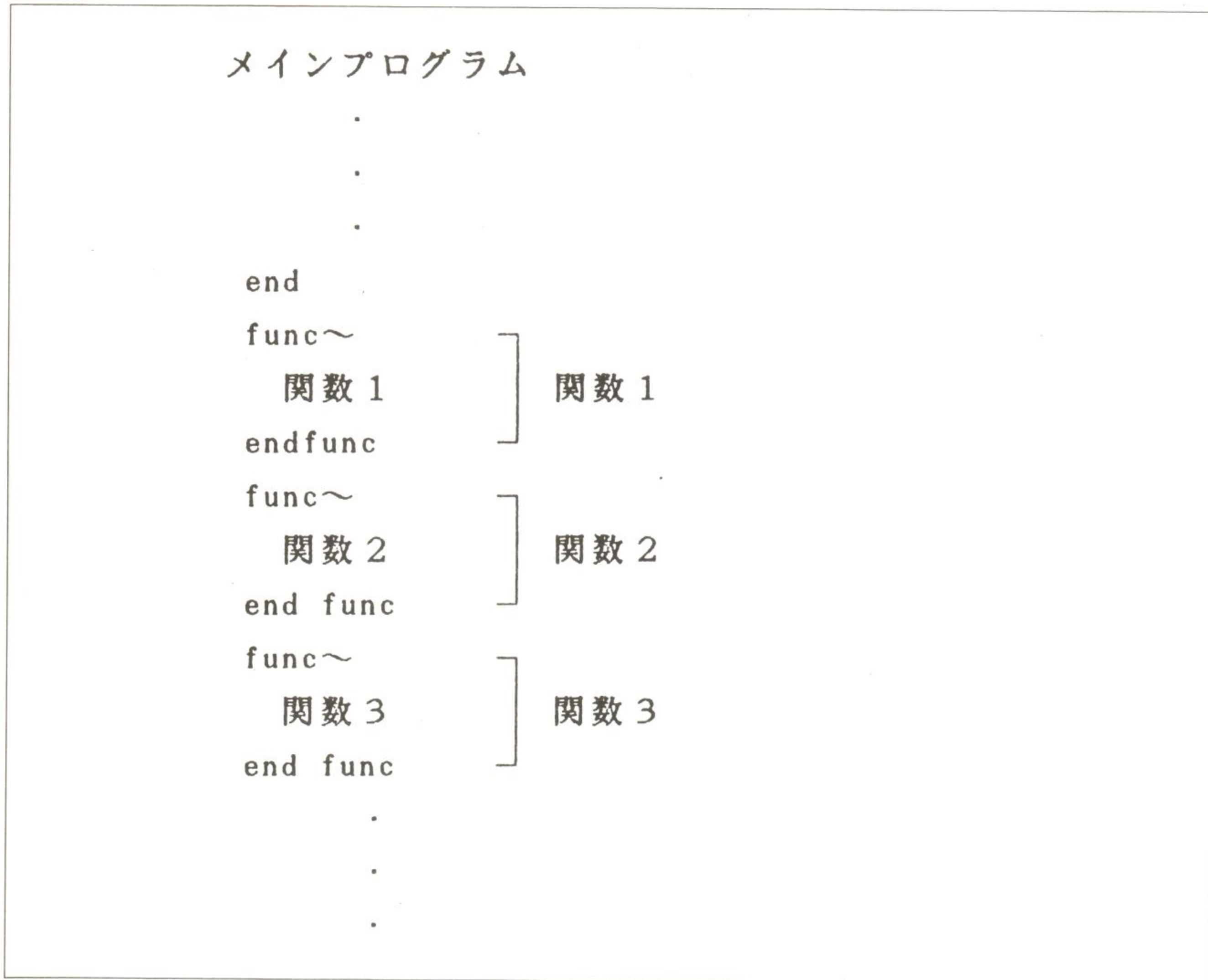
2-3-3 関数の型

関数には，引数と戻り値があります。これは２－３－１でも説明した通りですが，それぞれ引数，戻り値がどのような型の数値，または文字列等であるかは，これまで説明した通り変数の型をプログラマーが管理するBASICであるため，非常に重要なことになってきます。

標準関数及び外部関数では，マニュアルにその関数の引数と戻り値の型が説明されていますが，定義関数では，ユーザーがfunc ～ endfuncにより定義するので，その関数の引数，戻り値により型の宣言をしなければなりません。この場合，関数の型にあたるのは戻り値のことで，２－３－１で説明した通りに関数ｆ（ｘ）はそれ全体が戻り値であることから，関数ｆ（ｘ）の型というのは戻り値の型の宣言になります。引数の型宣言は，引数の前に一般の変数の型宣言同様に，四つの型のいずれかを使用しているものをおくことで完了するわけです。ただし，X-BASICでは引数に配列を使用することは，原則的に不可能であるため直接引数を配列とせず，配列変数はグロー

バル変数として，メインプログラムに置いて下さい。

ここで，X-BASICのプログラムを大きく二つにわけてみます。一つはメインプログラムで，プログラム全体の核となるもので，一応はこのメインプログラムでどんな事をするのかがつかめます。もう一つは関数で，ここでいう関数というのは定義関数のことで，X-BASICプログラムによりユーザーが作ったものをさしましす。X-BASICでは，定義関数はendステートメント（プログラム終了）以降に関数を置かなければならないという約束があります。したがって，全体の配置を図示すると**第2－27図**のような形になります。

```
メインプログラム
        ・
        ・
        ・
  end
  func～        ┐
    関数 1      │ 関数 1
  endfunc       ┘
  func～        ┐
    関数 2      │ 関数 2
  end func      ┘
  func～        ┐
    関数 3      │ 関数 3
  end func      ┘
        ・
        ・
        ・
```

第2－27図 X-BASICプログラムの構成

具体例として，これまでに出てきたステートメントだけで定義関数を使用して復習してみます。

```
10 int x=10, y=15, z
20 print x
30 print y
40 z=tasizan(x,y)
```

```
50 print z
60 end
100 func int tasizan(x;int,y;int)
110 return(x+y)
120 endfunc
```

第２－28図　定義関数サンプル例

ここで，新たにｚという変数を宣言していますが，これは tasizan（　）という関数の戻り値を格納する変数で，特にｚを宣言せずに50行で print tasizan（ｘ，ｙ）として良いのですが，ｙ＝ｆ（ｘ）の型にした方がわかりやすいのでｚに代入しました。

さて，このプログラムの実行結果は，**第２－29図**の様になります。

```
RUN
10
15
25
OK
█
```

第２－29図　プログラム実行

プログラムの流れを最初から説明していきましょう。10行目は各変数の宣言で，ｘは初期値を10としｙは15として，ｚには初期値がないので自動的に０になります。20，30行目はｘとｙを画面へ表示しています。さて，40行目からはここでのメインイベントで，tasizan（　）という関数の戻り値を代入しています。代入される戻り値，即ち tasizan（　）は，どのようなものかというと，引数が二つありそれぞれｘ，ｙが与えられています。このように，引数はいくつあっても構わないので，自分で必要な情報を引数として，関数に与えて複雑な仕事を関数に実行させることができます。ここでは，ただ二つの引数を加算して，その結果を戻り値として返すだけです。その関数の内容は，100行目～120行目で定義されています。定義関数は，func ～ endfunc で囲まれた部分ということになりますから，この関数の作業は110行目の return（ｘ＋ｙ）ということになります。ちなみに，100行目の関数の型宣言と引数の型宣言はご覧の通りです。return（　）というのは，関数内だけに存在するステートメントで，（　）内の値を戻り値にするということです。10＋15の25が tasizan（ｘ，ｙ）の戻り値なので，ｚ＝25ということになります。

2-3-4 関数と変数

２－２－４で紹介したグローバル変数とローカル変数ですが，これは関数と非常に深い関係があります。**第２－30図**を見て下さい。

```
10 int a=5,x
20 x=f(a)
30 print x
40 end
100 func int f(c;int)
110 int b
120 b=c-1
130 return(b*b)
140 endfunc
```

第２－30図　グローバル数とローカル数

このプログラムは，グローバル変数とローカル変数の両方を使用しています。10行から40行のendまでがメインプログラムで，100行目以降が定義関数です。メインプログラム中のａとｘがグローバル変数となり関数内のｂとｃがローカル変数となります。

ここで関数を定義する時に注意しなければならないのは，このプログラム中の変数ａとｃは引数の渡す側と渡される側の関係で，同一の変数ではないということです。このｆ（ｘ）という関数は引数が一つですから，カンコ内の数値または変数は一つしかありません。渡す側と，渡される側とで間違えることはほとんどありません。複数の引数になったときは，カッコ内の引数の順番が重要な情報になります。渡す側と渡される側の引数で重要なことは，変数に使用したアルファベットではなく，引数の個数と順番なのです。このことから，引き数に使用したアルファベットは，メインプログラムと関数定義の時の変数と一致していても，一致していなくても構わないのです。

さて，このプログラムの実行結果は次の通りです。ａに５が代入されて，関数ｆ（　）に渡され関数の引数のｃに代入されます。関数内で，int型のローカル変数として宣言されたｂへｃの内容から１引いたものが代入され戻り値としてreturn（)の（　）内でｂが２乗されて戻ります。

```
run
16
ok
■
```

もう少しローカル変数とグローバル変数がわかりやすくなる様に今のプログラムを**第２－31図**

の様に変更します。

```
10 int a=5,x
20 x=f(a)
30 print x
40 end
100 func int f(x;int)
110 int a
120 a=x-1
130 return(a*a)
140 endfunc
run
16
█
```

第2－31図 変数を2関数に変更

このプログラムでは，先程4種類あった変数がaとxの2種類になっています。10行から40行までのメインプログラム内の変数aとxはグローバル変数で，100行から140行までの定義関数内のaとxはローカル変数で，お互いに変数名は同一ですが，違う変数です。これを時系列的に文章で説明すると，**第2－32図**のようになります。

10行	int型の変数でaとxを宣言してaには5を代入した。
20行	f( )の関数呼出しを，引数のa＝5として渡し，戻り値はxに代入される。
100行	f( )の関数をint型で宣言し引数xをint型で宣言する。xにはa＝5が代入されx＝5となる。
110行	関数内でローカル変数aがint型で宣言する。
120行	xから1引いたものをaに代入。
130行	aの2乗を戻り値として，関数を終了する。
30行	戻り値を代入したxを画面へ表示する。
40行	プログラムを終了。

第2－32図 第2－31図の説明

この例でもわかる様に，aとxという変数には2種類の変数が存在しています。ローカル変数は，関数内だけでしか存在出来ません。**第2－33図**のプログラムは，同一の変数名の値を見ることが出来るものです。

```
10 int a=5,x
```

```
20 x=f(a)
30 print a
40 print x
50 end
100 func int f(x;int)
110 int a=1
120 print a
130 return(x)
140 endfunc
run
1                    ･･･120行の結果
5                    ･･･30行の結果
5                    ･･･40行の結果
OK
█
```

第2－33図　同一の変数名の値を表示

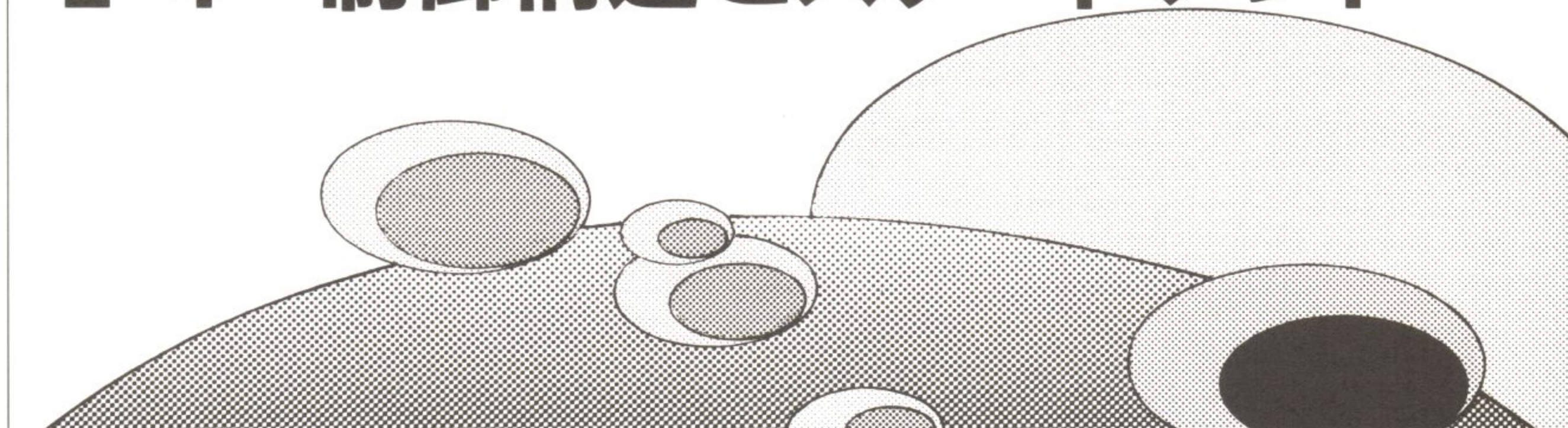
メインプログラム内のaは5を代入したのにもかかわらず関数内でa＝1としてprint文でも1を表示しています。メインプログラムに戻り，aをprintさせると5になっています。

さて，グローバル変数とローカル変数も少しずつわかってきたと思いますが，引数を持たない関数の場合についても解説してみます。一般的に関数というのは，引数に対し戻り値があってはじめて意味をもつのですが，X-BASICでは関数の呼び出しをすることに意味がある場合があります。このような場合には，その関数には，引数が必要なかったり，戻り値がない様な事があります（**第2－34図**）。

```
10 int x=10
20 f()
30 print x
40 end
100 func f()
110 print "関数1"
120 endfunc
```

第2－34図　引数を持たない関数の場合

この例では，従来のBASICでいうサブルーチンのようなもので，print "関数1"という処理をメインプログラムで実行しないで，関数で処理させています。もちろんこんな場合には，return（　）はなく，endfuncにより関数を抜け出します。

2-4　制御構造とステートメント

従来のBASICでは，関数の扱いが曖昧であったため，ほとんどがメインプログラムとサブルーチンの関係で処理をしていました。その核となるのがステートメントです。ステートメントというのはプログラムを構成するもので，各種の命令群です。X-BASICでは，この命令群の代わりに関数で各処理を実行します。そのためか従来のBASICでいうステートメントの数が減っています。その中に，これまであまりお目にかからなかった制御構造が加わってきました。制御構造というのはプログラムの流れを変えるもので，プログラムでは重要な地位を占めています。ここでは，この制御構造とステートメントについて解説します。

2-4-1　制御構造

BASICのプログラムは，行番号の小さい順にプログラムが実行されます。これは，X-BASICでも同じです。この行番号の小さい順という約束を破る役目をするのが制御構造なのです。従来のBASICでは，行番号の順序を変えるためgoto～，gosub～文が使われてきました。goto～，gosub～で必要とする情報は飛んで行く先の行番号です。これでは，行番号に依存するプログラムになってしまいます。X-BASICではこれらのgoto～，gosub～を使用せずにプログラミング出来る様に，次のような制御構造命令があります。

a）if～then～else
b）for～next
c）while～endwhile
d）repeat～until～
e）switch～case～default～endswitch

a）とe）はプログラムの流れを変えるもので，b）～d）は繰り返し作業などに使用されるループの命令です。

a）if（条件式）then（処理１）else（処理２）

これは，条件式が正しければ（真ならば）処理１を，間違っていれば（偽ならば）処理２を，実行します。elseが無い場合には，処理２も無くなり次の行へ処理が移ります。この制御構造命令

は従来のBASICにもあり，その場合には処理1や処理2の所で，goto～，gosub～と処理を移していました。しかし，X-BASICではgoto，gosubを使用しないで済むような工夫がされています。もともとgoto，gosubで他の場所に処理を移すほうが不自然で，その場所に十分処理をこなせるスペースがあれば，何も違う場所へ行ってしなくてもいい訳です。そこで，X-BASICでは変数の初期化でも使用した，｛ ｝（中カッコ）を使います。この｛ ｝（中カッコ）は，｛と｝で別な行にまたがっていてもいいことから，**第2－35図**の様な使い方が出来ます。

```
10 int x=15,y=10
20 if x>y then {z=x-y
30               z=z+1
40               print z
50              } else{z=y-x
60                     z=z+1
70                     print z
80                    }
90 end
run
6
ok
█
```

第2―35図　｛ ｝で複数行にまたがった処理を指定できる

これは，20行目の後半にある｛から50行目の先頭の｝迄が処理1で，50行目後半の｛から80行目の｝が処理2となります。このプログラムは，xとyの大小関係からその条件にあった計算をさせるプログラムです。このプログラムでは，｛ ｝の中が3行なので比較的楽に解釈できますが，もっと複雑になった場合は関数によりif～then～elseの中をすっきりさせます。

この辺がわかってくるとX-BASICのプログラムも覚えるのが早くなってきます。

if～then～elseの小さいスペースへ｛ ｝（中カッコ）を使用して全ての処理をすませてしまうことが，if～then～elseに限らずどの制御構造にもあてはまる重要なことです。

b）for（処理）next

ある処理を繰り返し実行しなければならないような場合，ループの制御構造を使います。前述のようにループの制御構造は3種類あり，このfor～nextはループする回数がわかっている様な場合に使用します。このfor～nextによるループは従来のBASICにもあって，よく知っている人もいるかと思います。しかし，この制御構造は従来のBASICでも行番号に管理されない制御構造であ

ったことに気が付いた方は少ないと思います。for～nextは，

　　for　初期値　to　終了値
　　処理
　　next

このように，forからnextまで囲まれた処理を，（終了値－初期値）の回数だけループします。実際のプログラムは**第２－36図**のようになります。

```
10 int x
20 for x=1 to 10
30 print x
40 next
50 end
```

第２－36図　for～nextサンプルプログラム

これは，第１章でも出てきたfor～nextの使用法でも一番簡単な使い方です。30行のprint文が１から10まで変数xを変化しながら繰り返します。したがって，１行のprint文も10行分の仕事をしたことになります。このプログラムの実行結果は**第２－37図**の通りで，30行目の文が10回実行されたことがわかります。

```
run
1
2
3
4
5
6
7
8
9
10
ok
```

第２－37図　実行結果

このfor～nextによるループは非常に単純にループを作ることが出来るため，たいへん面白い使い方をすることがあります。あるプログラムで少しの間プログラムに止まっていてもらって，現

在表示中のコメントをみてもらいたい様な場合，次のようにループ回数を多目にしてプログラムのウェイトをかけたりします。

　　　　表示処理
行番号　for x ＝ 0　to 1000：next
　　　　表示を消す処理

ウェイトする時間は終了値の1000を少なくすれば短くなり，多くすれば長くなります。

このfor～nextによるループで注意しなければならないのは，for～nextの初期値を代入する変数はint型でなければなりません。また従来のBASICではループする際のカウント幅が自由に設定できたのに対し，一つしかカウントアップ出来ません。従って一つずつカウントダウンしたい様な場合は，**第２－38図**のような演算をしなければなりません。

```
10 int x,y
20 for x=0 to 5
30 y=5-x
40 print y
50 next
60 end
run
5
4
3
2
1
0
ok
█
```

第２－38図　初期値を一つずつ減らしていく場合

また二つずつ増やす場合は，**第２－39図**のようになります。

```
10 int x,y
20 for x=0 to 5
30 y=2*x
40 print y
50 next
```

```
60 end
run
0
2
4
6
8
10
ok
█
```

第2—39図　初期値を二つずつ増やす場合

この様に一つずつしかカウントアップ出来ないために色々な工夫を要求されますので，みなさんも自分自身で色々確認してみる必要があります。

これまでは基本的なことでしたが，実際にはこんな簡単なループばかりではありません。for～nextの外側にfor～nextがある様な2重ループや，またその外にfor～nextがある3重ループなどが必要に迫られて出てくる事もあります。こんな場合，あわてずに一番内側のループから理解すれば少しずつわかってくると思います。

```
10 int x,y,z
20 for x=0 to 1
30     for y=0 to 1
40         for z=0 to 1
50             print x*100+y*10+z
60         next
70     next
80 next
90 end
run
0
1
10
11
100
```

```
101
110
111
ok
█
```

第２—40図　for～nextの三重ループ

なにやら２進法のような感じがします。これは，三つの変数が順に０と１に変化しながら，一番外側の変数が100倍，二番目の変数は10倍、三番目の変数はそのままで加えたものを表示しています。ループを重ねることで，こんな面白い事も出来ます。なお，このプログラムで，ループが内側になるほどステーメントを書く位置が後ろになっていますが，これはインテントといって処理の深さを表しています。今どの位置にいてどんな仕事をしているかなどが，記述してある位置によってわかりますから非常に見やすいプログラムになります。この場合も，深さに依って今どのループにいるかなどが表面に出てきます。

c）while（条件式）endwhite

ここからは，今までのBASICではあまりみかけない制御構造です。この命令もループする制御構造ですが，条件式が満たされている時だけ繰り返します。したがってループしながら段々情報が変化していくようなループで使用します。基本動作はまず最初に条件式で，ループ内の処理をするかループせずに次へ行くかを判定します。条件式での判定は，if～then～elseと同様に条件式が正しければループ内の処理をし、正しくなければendwhile以降へ処理が移ります。**第２—41図**のプログラムで確認して下さい。

```
10 int x=5
20 while x<10
30 print x
40 x=x+1
50 endwhile
60`end
run
5
6
7
8
9
```

```
ok
```

第２—41図　while〜endwhileサンプルプログラム

変数宣言のときに５を代入した変数xはwhileに入るところで，条件x＜10で判定されますが，xは10より小さいのでwhile内に入ります。30行目でxが表示され，40行でxが１加算され20行のwhileの所へもどります。こうしてxが一つずつ加算されていって10になり20行のwhileへ戻った時，条件式はx＜10を間違えと判定します。したがってwhile内には入らずendwhileの次の行60行のendへ進みプログラムを終了します。

d）repeat（処理）until（条件式）

これはwhile〜endwhileと非常によく似たループですが，条件判定がループの終わりにあり，また条件が満たされるまでループをします。したがって，while〜endwhileでは条件次第では一度もループ内部を通らない事もあったのですが，repeat〜untilでは最低一回は，ループ内を通過します。

第２—42図のプログラムで条件式が満たされるまでループすることを確認して下さい。

```
10 int x=5
20 repeat
30 print x
40 x=x+1
50 until x>10      ・・・・・判定が逆になっている事に注意して下さい。
60 end
run
5
6
7
8
9
10
Ok
█
```

第２—42図　repeat〜until〜サンプルプログラム

このように，whileとrepeat文を使い分けることでより効率の良いプログラムを記述することが出来ます。

さて，ここでwhile文とrepeat文の面白い使い方を紹介します。従来のBASICでは無限ループを作る場合gotoを使い前の行へ処理を戻しましたが，X-BASICではいま紹介したwhile文やrepeat文を使う事が出来ます。while，repeatで無限ループを作るというのは，それぞれの条件式が何回

ループしても満たされなければ良いのです（while文では常に真，repeat文では常に偽）。

```
while 1                  repeat
  ⋮                        ⋮
処理                     処理
  ⋮                        ⋮
endwhile                 until 0
```

このように記述をすると処理は無限に繰り返されます。**第2－43図**のプログラムを実行してみて下さい。

```
10 while 1
20 print "無限ループ"
30 endwhile
40 end
run
無限ループ
無限ループ
無限ループ
無限ループ
無限ループ
無限ループ
  ⋮
  ⋮
```

第2－43図　無限ループプログラム

このプログラムを停止する時は，BREAKキーを押して止めて下さい。この無限ループでの条件式が1というのは，X-BASICが条件式を正しいか正しくないかを0か0でないかに依って行っているためで，条件式の代わりに1と置いてしまうことで判定が常に正しいと解釈されるためです。

e）switch（条件式）

```
case：処理1
case：処理2
case：処理3
default：処理n
endswitch
```

何やら今までに見たこともないような制御構造です。a）のif文と同じ分類になる制御構造で，

処理の分岐をするものです。if文と違うところは，if文のように処理が二つに分岐するのでなく，caseに続く式や値とswitchにある条件式とを比較し，一致している場合にそのcaseに続く処理を実行します。**第2－44図**のプログラムで動作を確認して下さい。

```
10 int x=3
20 switch x
30 case 1:print "apple"
40 case 2:print "orange"
50 case 3:print "lemon"
60 endswitch
70 end
run
lemon
ok
```

第2—44図 switch文サンプルプログラム

このプログラムでは，xという変数に初期値3を代入してswitch文に突入します。switch文での条件式はxなので，このxとそれぞれのcaseの式とが順番（行番号順に）比較されて一致したものについて，そのcase文が実行されます。したがって，x＝3ですからcase 3：で一致しますからprint "lemon" が実行されます。このように，いくつかの処理を条件により分岐させる役目をします。しかし，使い方に若干注意が必要です。今のプログラムで，**第2－45図**の様に初期値を1にして実行してみます。

```
10 int x=1
20 switch x
30 case 1:print "apple"
40 case 2:print "orange"
50 case 3:print "lemon"
60 endswitch
70 end
run
apple
orange
lemon
```

```
ok
█
```

第2―45図　初期値を1にすると

この様に，最初のcaseで判定が一致してappleと表示されるところまではいいのですが，次のcaseとその次のcaseも実行されてしまいました。switch文は，一度正しいと判定されるとその後のcaseを無条件で通過してしまいます。また，今の例では三つに分岐しましたが，どれにもあてはまらなかった時の分岐として，defaultがあります。この働きをみるために，今のプログラムの初期値を4にして55行を追加して下さい（第2―46図）。

```
10 int x=4
20 switch x
30 case 1:print "apple"
40 case 2:print "orange"
50 case 3:print "lemon"
55 default:print "fruit"
60 endswitch
70 end
run
fruit
ok
█
```

第2―46図　defaultの実行

この様にdefaultはどのcaseでも引っ掛からなかったものを全て引き受けてくれる役目をします。この場合も，先に説明したひとつ目のcaseを実行した時にいもづる式にその後のcaseを実行してしまうのと同じ様にdefaultも実行してしまいます。

f) break と continue

条件にあったcaseだけを実行するにはどうしたらいいのでしょう。ここで，breakという制御構造を**制御**する命令を使います。breakは現在実行している制御構造を中止して，制御構造を抜けて次の処理へ移るというものです。先のプログラムにbreakを入れて条件にあったものだけを実行出来る様にしたものが，**第2―47図**のプログラムです。

```
10 int x=1
20 switch x
```

```
30 case 1:print "apple":break
40 case 2:print "orange":break
50 case 3:print "lemon":break
55 default:print "fruit"
60 endswitch
70 end
run
apple
ok
■
```

第2—47図　breakで条件にあったcaseだけを実行

この他にbreakはfor～next, while～endwhile, repeat～untilにも有効です。これらはループですから，breakがあるとループを抜け出しループ以降の処理へ移ります。しかし，このbreakはむやみに入れてあるのではありません。だいたいはif文とペアで使用されます。

```
10 int x=1
20 while 1
30 if x=5 then print "終わり":break
40 print x
50 x=x+1
60 endwhile
70 end
run
1
2
3
4
終わり
ok
■
```

第2—48図　if文とbreakの組み合わせ

これは，変数xの初期値が１でwhileの無限ループへ入り，xが５になるまでループして５になった所でif文により〝終わり〟と表示しbreakによりループを抜けます。この様に，break文の使用には色々な判定をした結果としてbreakを実行するようになります。

breakは実行されると，その後ろにあるループ処理を一切実行せずループを抜ける働きがありますが，これとよく似た命令で，continueがあります。この命令も制御構造を制御するもので，continueが実行されるとそれ以降のループ処理をせずにループの最後に飛んでいきます。ループを続けていき条件にあったものだけを処理する様な時に使用します。したがって，このcontinueはfor～next, while～endwhile, repeat～untilの３種類にのみ使用可能です。ちなみにbreakはswitch～case～default～endswitchを加えた４種類に使用可能です（**第２—49図**）。

```
10 int x
20 for x=0 to 10
30 if x<5 then print "前半":continue
40 print "後半"
50 next
run
前半
前半
前半
前半
前半
後半
後半
後半
後半
後半
後半
ok
█
```

第２—49図　continueサンプルプログラム

変数xが，５よりも小さい時はif文により〝前半〟と表示され，同時にcontinueも実行しますからforループの最初へ戻りまた繰り返されます。continueを実行した時には，for～nextの場合は変数が一つ増加してループの最初へ戻り，while～endwhile, repeat～untilの場合は条件式が判定されます。こうしてxが５を越えるとif文により条件が不成立となりcontinueは実行されず，40行のprint〝後半〟が実行されて，変数xが11になった時にfor～nextの終了値を越えてループ処理を終わります。

このbreak, continueの使用により，前述の制御構造命令と共により一層複雑な処理が可能になります。行番号にとらわれる事なくX-BASICの処理を進める大切な制御構造ですから十分理解してください。

2-4-3 ステートメント

従来のBASICでは処理の中心的存在だったステートメントですが，X-BASICではほとんどの処理が関数化されてしまったため，ステートメントとして分類されているのは次の31個しかありません。

	BEEP	◎	IF THEN ELSE
◎	BREAK		INPUT
◎	CHAR	◎	INT
	CLS		KEY
	COLOR		LINPUT
	COLOR[　]		LOCATE
	CONSOLE		PRINT
◎	CONTINUE		REM
	DIM	◎	REPEAT UNTIL
	END		SCREEN
	ERROR ON/OFF		STOP
	EXIT（　）	◎	STR
◎	FLOAT	◎	SWICH CASE DEFAULT END SWICH
◎	FOR NEXT	◎	WHILE END WHICH
◎	FUNC ENDFUNC RETURN（　）		WIDTH
	GOSUB RETURN		
	GOTO		

この中で，本書の中で既に紹介されているものが，◎印のついたものです。この項では◎印のついていないものについて説明します。

（a）beep

Human68K（OS）が起動された時（電源投入時やリセットした時）に，CONFIG.SYSというファイルの中で定義されているBEEP.SYSという名前のファイルに入っているADPCMの音声データを再生します。簡単にいえば，「ピー」とスピーカーから音が出るのです。しかし，X68000ではADPCMの音声データであることから，ADPCMによる録音された音声データ（例えば「エラーです」。など）を，BEEP.SYSとしておけば，X-BASICでプログラム中にエラーが起きた時など，このBEEP命令で「エラーです。」などのメッセージを再生することも可能です。

（b）cls

テキスト画面のクリアーをします。X-BASICでのディスプレイ画面は，テキスト，グラフィックス，スプライトの３種類の画面の合成ですから，その内のテキスト画面だけをクリアするわけです（テキスト画面というのは，文字を出力する画面のことです）。

（c）color 属性

テキスト画面の文字の属性を設定する命令です。文字の属性というのは，文字の色，標準か強調か，標準か反転かを意味しています。色には，黒，シアン，黄色，白の４種類があり，その他の２種類（標準↔強調と標準↔反転）については２種ずつのため４×２×２の16通りの設定ができます。設定方法は次の通りです。

	color	n	n：0～15		
0	標準文字	黒	8	反転文字	黒
1	標準文字	シアン	9	反転文字	シアン
2	標準文字	黄色	10	反転文字	黄色
3	標準文字	白	11	反転文字	白
4	強調文字	黒	12	反転強調	黒
5	強調文字	シアン	13	反転強調	シアン
6	強調文字	黄色	14	反転強調	黄色
7	強調文字	白	15	反転強調	白

それでは，実際に**第２－50図**で動作を確認してみます。

```
10 int x
20 for x=0 to 15
30 color x
40 print "X68000"
50 next
60 color 3
70 end
```

第２－50図　color文サンプルプログラム

このプログラムでは，X68000という文字が色々な色や，字体に代わりながら表示していきます。

（d）color　[　]

テキストのパレットの色を変更します。X68000のテキストに使用できる色は，65536色中４色あります。（c）のcolor命令のところでの４色は，黒，シアン，黄色，白の４色でしたが，それぞれの色を65536色のカラーコードの中から選び設定することができます。[　] の中の数値は４個あり，それぞれが，０～65536までの数値となります。設定方法は次のようになります。

```
color [a, b, c, d]
 aは,(c)のcolor 命令の時の0, 4, 8, 12の色
 bは,(c)のcolor 命令の時の1, 5, 9, 13の色
 cは,(c)のcolor 命令の時の2, 6, 10, 14の色
 dは,(c)のcolor 命令の時の3, 7, 11, 15の色
```

(注) カラーコードとパレットコードについては第3章のGRAPH. FNCの項を参照して下さい。

(e) console

テキスト画面は文字を表示する画面で, 768×512ドットの画面では96文字×32行, 512×512ドットの画面では64文字×32行の表示ができることになります(console命令は, 256×256ドットの画面では使用できません)。

どちらも32行の表示ができますが, 最下行から次の行に表示が移る時に, 画面が1行ずつ上へずれます。これをスクロールといいますが, 何も指定していない時は全画面がスクロールします。このスクロール機能を, 何行目から何行分に限定するかを指定するのがconsole命令です。したがって, スクロール開始行の指定は, 0～30となりスクロール範囲は, 1～31になります。実際の設定方法は次のようになります。

```
console 開始行, 範囲, ファンクションキーの表示
```

(注) ファンクション表示分として最下行は, ユーザーが使用できません。また, 3番目の数値は, 0か1で, 0ならばファンクションキー表示なし, 1ならばファンクションキー表示ありとなります。

(f) error on/error off

一般的に, プログラム実行中にエラーが発生すると, プログラムが停止してどの行でどんなエラーが発生したかを画面に表示します。このエラーの中で, 関数によるエラーについては, エラーであることを戻り値で知らせてくるものもあり, また, プログラマーがエラーが起こる可能性を判断できているプログラムもあります。こんなときに, いちいちエラーが発生するたびにプログラムが停止していては困るので, 関数エラーが発生しても, そのまま次のステートメントへ進めるように, error offがあります。また, エラーの発生を知らせる様にするために, error onがあります。

(注) プログラムをRUNにより実行した直後は, 強制的にerror onの状態になります。

(g) exit ( )

( )(カッコ)がついて関数の形態をとっていますが, 引数も戻り値もないので, 命令(ステートメント)として分類されています。この命令は現在の状態を全て破棄できる状態(ファイルがオープンしていればクローズする)に処理をして, 親プロセスへ戻ります。親プロセスという

のは，X-BASICを起動した時の状態をいい，VSモードならVSモードへ，コマンドモードならコマンドモードへ戻ります。この命令を使用したプログラムを制作している時は，少し直してはディスクへ保存を繰り返すか，EXIT（　）をプログラム完成時に書く様にします。いったんこの命令を実行すると，プログラムエリアも破壊してしまいます。

(h) gosub／return

従来から引きずっている命令で，サブルーチンをコールします。飛ぶ先は，行番号ですからgosub 行番号　となり，サブルーチンでreturnを見つけると，gosubで飛んだ次のステートメントから処理がつづきます。とにかく使用できるということだけで使わない様にするのが一番です。

(i) goto

(h)同様，行番号を目標としてプログラムをジャンプします。この命令も使用しない様にした方が得策です。

(j) input　キーボードから入力されるデータ（文字，数字）を指定した変数へ代入します。この命令は使い方が非常に多い命令です。

(1)input a（aは入力したいデータの型に宣言されていなければならない）

この命令が実行されると画面には？（クェッションマーク）が表示されて，キーボードから入力待ち状態になります。入力するのは，代入される変数の型にあったものでないとデータの型が違うので，メッセージが出力されて再度入力待ち状態になります。文字列変数の場合には，入力される文字数にも注意して下さい。

(2)input〝メッセージ〞；a

〞（引用符）で囲んだメッセージを画面に出力し，そのメッセージの後ろに？（クェッションマーク）も出力され，入力待ち状態になります。あとは(1)と変わりません。

(3)input〝メッセージ〞，a

(2)と変わらないのですが，メッセージの後ろには何も出力されません。

(4)input a，b（aとbは，それぞれ入力したいデータの型で宣言されていなければならない）

複数のデータを入力する時に使用します。変数をカンマで区切って，並べて記述します。入力方法は，データとデータをカンマで区切って入力します。変数の個数より少く入力すると，〝データの個数が足りません〞とメッセージが出力されて入力待ち状態になります。また，データの個数が変数の個数より多く入力してしてしまったときは，〝データの個数が多すぎます。〞とメッセージが出力されて多かった分のデータは無視されて次へ進みます。

それでは**第2－51図**で，一般的な入力方法の確認をしてください。

```
10 int a
```

```
20 str b
30 input "数値",a
40 input "文字",b
50 print a
60 print b
70 end
```

第２—51図　input文サンプルプログラム

（k）key

テキスト画面の最下行にファンクションキーの内容が表示されていますが，そのファンクションキーの機能を定義します。KEYナンバーは１～20まであり，１～10がF１～F10に対応していて，11～20は[SHIFT]＋[F１]～[F10]に対応しています。文字列の長さは32文字まで有効です。設定方法は次の通りです。

10　key１，"renum　@M"

@MはコントロールMで，⏎（キャリッジリターン）の事です。このようにコントロール文字は，@を付けて記述します。

（l）linput

（j）input同様，キーボードからデータを入力する命令ですが，文字列変数しか指定できません。input文では，その機能上の制約で　，（カンマ）や　"（引用符）は入力できませんが，linputではキーボードから入力できる文字は，全て代入されます。ただし，入力できる文字数はstr型変数の宣言した時の文字数の制限がありますが，この他に画面の１行文の制限もありますので注意が必要です。実際の用法は次の通りです。

linput　"メッセージ"；変数（str型）

（m）locate

指定した位置へカーソルを移動します。その他の機能として，カーソル表示をする／しないの指定もできます（プログラム実行中のみ有効です)。

locate カーソルｘ座標，カーソルｙ座標，カーソルスイッチ

カーソルx座標は０からはじまり画面サイズにより，512×512の時は63，768×512の時は95となります。カーソルy座標は０～31となります。

第２—52図は，locateを使ったプログラムです。

```
10 cls
20 locate 31,16
```

```
30 print "X-BASIC"
40 end
```

第2―52図　locate文サンプルプログラム

(n) print

テキスト画面に文字列や数値を表示します。この命令には，いくつかの書式があります。文字列を表示する時は，str型に変数か，″（引用符）で囲んだ文字の並びをprintの後ろへ置きます。数値を表示する時は，数値型変数（char型，int型，float型）をprintの後ろへ置きます。色々と細かい書式がありますが，次のように分類されます。

（1）print　変数

変数の内容が画面に表示されます。この変数は，数値型変数（char型，int型，float型）の場合には数値が，文字型変数（str型）の場合には文字列がそれぞれ出力されます。この変数のことを出力される要素と呼びます。出力された後で，次に出力する位置（カーソル位置）が次の行の先頭になります。これを改行といいます。ここで間違え易いことに，変数名がそのまま出力されるように思いがちですが，変数名がそのまま画面に出力されるのではありません。

（2）print　〝文字の並び″

″（引用符）で囲まれた文字の並びの場合は，その文字の並びがそのまま画面へ出力されます。″（引用符）で囲まれた文字の並びは，str型変数の初期化でも出てきましたが，単純な解釈で文字列だと思えば（1）の文字列変数の出力であると理解出来ます。この場合も（1）同様，出力後のカーソルの位置は次の行の先頭になります。

（3）print　要素1；要素2；・・・；要素n

print文では，複数の要素を；（セミコロン）で区切り続けて記述することが出来ます。要素は，文字でも数値でもその混合でも構いません。画面にはこれらの要素が連続して出力されます。また，最後を要素で終わらず；（セミコロン）で終わると，(1)，(2）の様な改行はしません。この場合，次にprint文が実行されると現在のカーソル位置から出力されます。最後が要素で終わっていれば改行します。

（4）print　要素1，要素2，・・・，要素n

（3）と同様に要素を　，（カンマ）で区切ります。この場合，（3）では要素同士が連続して出力されるのに対し要素同士を8文字に区切って出力します。一つの要素が8文字以内であればあたかもその要素が8文字であったかのように、次の要素は8文字目から出力されます。もし，要素が8文字を越えるような場合には次の8文字の区切りになり，8文字なら次の要素は連続して出力されます。

，（カンマ）を最後に付けて終わると（3）同様，改行されずに次の出力されるのは最後に出力

（5）print using〝書式″；変数（数値型）または数字

書式部分に決められた書式を表す記号の並びを記述すると，変数または数値がその書式で表示されます。決められた書式を表す記号は，次の通りです。

＃（シャープ）	表示する数値の桁数を指定します。
．（ピリオド）	小数点の位置を指定します。はみ出した桁は4捨5入します。
＋／－	符号を付けて表示します。－の場合は，＃の後ろに設定します。
＊＊	空白を＊（アスタリスク）で埋めます。
￥￥	数字に￥（円マーク）を付けます。
＊＊￥	空白を＊（アスタリスク）で埋め，先頭に￥（円マーク）を付けます。
^^^^^	数値を指数を使った表現にします。
，（カンマ）	3桁毎に　，（カンマ）を付けて表現します。

（6）print using〝書式〞；変数（文字列型）または文字列

（5）と同様に，文字列について色々な書式を指定することが出来ます。

！	文字列の先頭の1文字だけを出力します。
&（空白）&	文字列を&から&までの文字数に指定します。
_（アンダーバー）	これは，（5），（6）で紹介した書式を設定する記号を _（アンダーバー）後の1文字だけをただの文字にするものです。

（注）これらの記号は，組み合わせて使用することが出来ます。色々と試してみて下さい。

o）rem（／＊）

remarks（注釈）の事で，プログラムに関係したコメントや覚え書きなどを注釈文としてプログラム中へ書き込むことが出来ます。1行全て注釈の場合はそのまま／＊に続いて書き込むことが出来ますが，ステートメントの後ろに注釈文を書き込む場合は，：（コロン）を付けて他のステートメント同様にマルチステートメントにしなければなりません。実際の使用例は**第2－53図**の通りです。

```
10 /*これは注釈文です。
20 print "test":/*注釈文のテストです。
```

第2－53図　rem（／＊）サンプルプログラム

p）screen

表示画面のサイズ，実画面のサイズ，解像度，グラフィックス画面のon／offを指定します。このステートメントの詳細は，第3章のGRAPH.FNCの所で解説します。

q）stop

プログラムの実行を中断します。これは，プログラマーが意識してプログラム中に入れておくもので，実行中に実行過程が良く判らないような所に入れ，そこでストップして途中の経過を確認する時などに使用します。一般的には，完成したプログラムにはあまり使われていません。このstopによって中断したプログラムを再開する時はcontコマンドを使用します。しかし，中断した時にプログラムの変更をすると，再開できなくなりますから注意して下さい。

r）width

この命令は，X-BASICのマニュアル上ではコマンドとして紹介されています。コマンドは，第1章でも説明しましたがプログラム中では使用できません。しかし，このwidthについては例外で，プログラム中の使用が許されています。従って，他のステートメントと同様にダイレクトモードで使用したり，プログラムモードで使用したりしてもかまいません。どちらかというと分類としては，ステートメントに属するものです。

この命令は，画面に表示される文字数を変更するもので，X-BASICでは96文字と64文字が許されています。ただし，画面に表示される行数に変化はありません。**第2－54図**は，このwidthを実際にプログラム中に使用した例です。

```
 10 int x
 20 width 96
 30 locate 48,16
 40 print "WIDTH 96"
 50 for x=0 to 2000:next
 60 width 64
 70 locate 32,16
 80 print "WIDTH 64"
 90 for x=0 to 2000:next
100 width 96
110 print "WIDTH 96 AGAIN"
120 for x=0 to 2000:next
130 width 64
140 end
```

第2－54図　widthサンプルプログラム

このプログラム結果は，表示画面を96文字モードに切り替え，画面中央に〝WIDTH 96〟と表

示して，しばらくたってから画面モードを64文字にして画面中央に〝WIDTH 64〞と表示して，またしばらくたってから画面モードを96文字して画面中央に〝WIDTH 96 AGAIN〞と表示します。そして，またしばらくたつと，元の64文字の状態に戻ります。

このプログラムを実行してみると，cls命令がないのに画面モード切り替えの時に表示画面がクリアされています。この命令を実行するとテキスト画面がクリアされることに注意して下さい。またこの例では確認出来ませんでしたが，この命令を実行するとテキスト画面がクリアされるだけでなく，グラフィック画面も同時にクリアされてしまいますので注意して使用しないとせっかく書いたグラフィックスの絵が一瞬のうちに消え去ってしまう事故も起こりかねません。ところで，今の例で96文字と64文字しか指定しなかったのは，この命令に使用出来る数字は64と96しかないからで，ステートメントの文法説明としては，

```
width 64
width 96
```

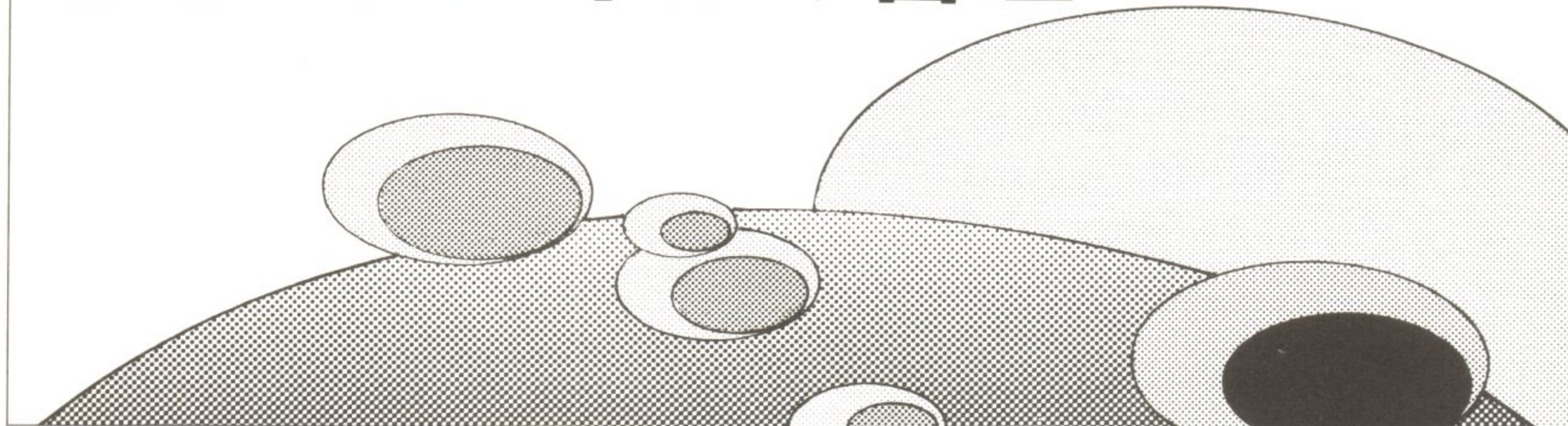
の２種類の命令があると覚えてもらえば結構です。

さて，**第２−54図**の中にあまり意味の無さそうな行が３行ほどあります。

```
for x＝0 to 2000：next
```

これは，制御構造のところで紹介したプログラムのウエイトです。このプログラムでは，表示画面が96文字モードや64文字モードに切り替わった時にそのことを確認するために，print文で現在の表示モードで文字を出力しますが，出力後すぐ次の表示モードに切り替わってしまうので，まったくprint文による表示を見ることが出来ません。そこで，このforループによるウエイト（空ループ）で一定の時間，表示している内容を見ることが出来るのです。

これまで色々と説明してきたステートメントは，サンプルプログラムでは体験できない良いところや，なかなか使う時に厄介なものなどがたくさんあります。たえまない探究心があれば乗り切れますし，また新しい使い方なども覚えます。常に実行してみて疑問を持つことが必要です。

2-5　ファイルの管理

これまでは，プログラム作成に関することを解説してきました。プログラムは前述のように，X68000のメモリ上（X-BASICのプログラムメモリ）に書き込まれています。このままでは，電源を切ってしまえばプログラムメモリ上のプログラムは消滅してしまいます。X68000には，2HDタイプのフロッピーディスクが2機内蔵されています。X-BASICでは，このディスクを利用してプログラムを保存したり，保存してあるプログラムをプログラムメモリへ呼び出したりすることが出来ます。

2-5-1　ファイルの保存

X-BASICのプログラムをディスクへ保存する時には，現在作成中または完成したプログラムに名前を付けて保存します。この名前のことをファイルネームと言い，保存された後はHuman68kで管理されているファイルと共有のものになります。一般的にX-BASIC起動時には，〝BASIC〟というディレクトリ内にいるので，このディレクトリ内に保存する場合には，ディレクトリ名などのわずらわしい操作は必要ありません。本書では，OSの解説書ではないため基本的なファイル操作は，〝BASIC〟ディレクトリ内だけに限定しています。

さて，プログラムの保存は，saveというコマンドにより実行します。このコマンドの書式は，次の通りです。

```
save "ファイルネーム"
```

このファイルネームは8文字まで，その後ろに　.　（ピリオド）を付けて3文字の拡張子を付けることが出来ます。X-BASICのプログラムに付ける拡張子はBASまたはbasと決められています。また，saveのコマンドでプログラムを保存する時にこの拡張子を省略すると自動的に.BASの拡張子が付きます。簡単なプログラムを作成し実際にそのプログラムを保存してみます（**第2－55図**）。

```
10 int x=10,y=15,z
```

```
20 z=x+y
30 print "z=";z,"x=";x,"y=";y
40 end
run
z=25     x=10     y=15
OK
save "test"
OK
█
```

第2—55図 "test" というファイル名でsave

この例では，一度プログラムを実行してから〝test〟というファイルネームでディスクへ保存しています。保存されたことを確認するために，filesコマンドを実行すると，**第2—56図**の様になります。

```
files
  496 Kバイトが使用可能です
     "A:¥BASIC¥BASIC              .X  "
                                   /*    46290 87/05/15 12:00:00
     "A:¥BASIC¥BASIC              .CNF"
                                   /*      142 87/05/15 12:00:00
     "A:¥BASIC¥AUDIO              .FNC"
                                   /*      540 87/05/15 12:00:00
     "A:¥BASIC¥GRAPH              .FNC"
                                   /*    35294 87/05/15 12:00:00
     "A:¥BASIC¥MOUSE              .FNC"
                                   /*      926 87/05/15 12:00:00
     "A:¥BASIC¥MUSIC              .FNC"
                                   /*    21528 87/05/15 12:00:00
     "A:¥BASIC¥SPRITE             .FNC"
                                   /*     2656 87/05/15 12:00:00
     "A:¥BASIC¥STICK              .FNC"
                                   /*      352 87/05/15 12:00:00
     "A:¥BASIC¥test               .bas"
                                   /*       82 87/10/26 14:42:36
```

```
OK
■
```

第２−56図　filesで確認

“BASIC”ディレクトリ内に保存されているファイル名がいくつか表示されます。この中に今保存した“teat.bas”と言うファイルネームがあり，拡張子の．basも付いています。この他に，その下にはこのファイルのサイズやファイルを保存した日付や時間が表示されています。

2-5-2 ファイルの呼び出し

ディスクへ保存したプログラムは，必要な時にプログラムメモリへ呼び出すことが出来ます。プログラムをプログラムメモリへ呼び出すには，loadコマンドを使用します。loadコマンドの書式は次の通りです。

```
load ”ファイルネーム”
```

このloadコマンドに付けるファイルネームにも拡張子を省略することが出来ます。２−５−１で保存したプログラムを第２−57図の様に，実際に呼び出してみます。

```
new
OK
list
OK
load"test"
OK
list
10 int x=10,y=15,z
20 z=x+y
30 print "z=";z,"x=";x,"y=";y
40 end

OK
■
```

第２−57図　ファイルの呼び出し

ご覧のように，newした後のlistでもプログラムメモリ内にはもうプログラムが残っていない事

が確認できます。この後で，loadコマンドを実行しlistしてみると，今度はプログラムがあります。今の例では，loadコマンド実行前にnewをしていますが，実際にはloadコマンドを実行するとプログラムメモリ内は一度消去されますので必要ないのですが，逆に消したくないプログラムも消えてしまいますから気を付けて下さい。

2-5-3　呼び出し後の自動実行

X－BASICのコマンドの中には，プログラムを呼び出した直後に実行することが出来るものがあります。これまでも何度となく出てきましたが，runというコマンドがそれです。使用方法は，loadコマンドと全く同じです。実際には，先程のloadをrunに置き代えたものを実行してみて下さい。

```
run "test"
```

2-5-4　プログラムの結合

第2章のコマンド説明の中に出てきましたが，load@，save@というコマンドを使用してプログラムの結合をします。ここでは，**第2－58図**のプログラム(二つ)を結合する手順を解説します。

プログラム1

```
10 print "X-BASIC"
20 print "is NO.1"
30 end
```

プログラム2

```
10 print "X68000"
20 print "is NO.1"
30 end
```

第2－58図　2本のサンプルプログラム

手順

(1) **プログラム1**をキーボードから打ち込む。
(2) このプログラムをsaveコマンドで保存する。
(3) newコマンドでプログラム1を消去する。
(4) **プログラム2**をキーボードから打ち込む。
(5) このプログラムをsava@コマンドで保存する。
(6) loadコマンドで，〝test 1〟を呼び出す。
(7) 続いて，このままの状態でload@コマンドを次の書式で結合します。
load@ "testl" ,40,10
(8) この状態でlistして**第2−59図**のようになったか確認します。

```
list
10 print "X-BASIC"
20 print "is NO.1"
30 end
40 print "X68000"
50 print "is NO.1"
60 end
OK
█
```

第2—59図　2本のプログラムが1本に

ご覧の様に，test 1 ,test 2 とが結合されています。プログラムが結合した後の内容としては，30行目のendは必要ないのでdeleteまたは1行削除の方法で削除する必要があります。また，このように結合したプログラムを保存するのを忘れない様にして下さい。

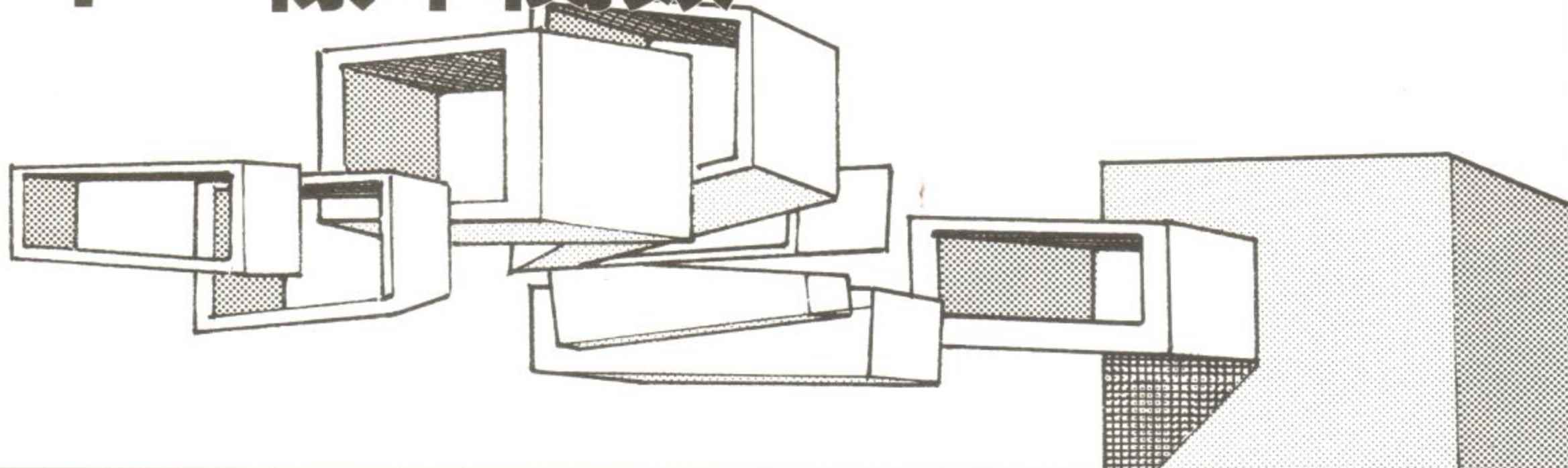
さて，save@コマンドとload@コマンドの書式ですが，save@については，saveコマンドと全く同じですが，load@コマンドは少々違います。

load@ "ファイルネーム"，(結合を開始する行番号)，(増分)

先程の例では，test 2 を40行目から結合し，10行間隔で結合したことになります。

第3章

標準関数と外部関数

3-1　標準関数

X-BASICには，X-BASIC本体に内蔵された関数が何グループかあります。これらの関数は，パーソナルコンピュータにとって基本となるファイルの入出力に関するものとか，数値や文字の処理とか，データの変換とかいうもので，何の定義もなしにいつでも使用することが出来ます。この関数群のことを**標準関数**といい，**第3－1図**の様に分類されます。

(1)ファイル入出力関数
(2)データ変換関数
(3)文字列処理関数
(4)数値演算関数

第3－1図　標準関数

これらの関数群の内容は，色々に分類することが出来ます。従って，ここからの説明は単なる機能だけでなく引数や戻り値の分類等によっても説明を分けていきます。

3-1-1　ファイル入出力関数

一般にファイルというと，マイコン，パソコンを少し知っている方なら大抵はカセットテープレコーダやフロッピーディスクを思い浮かべます。もちろん外部記憶装置としてのそれは，世の中ではよく知られていますが，Human68kにおいては色々なデバイスをサポートしています(**第3－2図**)。

- コンソール（キーボードとディスプレイ）
- RS-232C（シリアルインターフェース）
- プリンタ
- フロッピーディスク
- ハードディスク
- ADPCM

第3－2図　Human68kでサポートしている各種デバイス

当然の様にHuman68kがこれらのデバイスをサポートしていれば，X-BASICでも使用可能です。従って，これから解説する関数群に付いてもこれまでの認識は改めておく必要があります。しかし，ここではフロッピーディスクを対象とした解説とします。

さて，このファイル入出力関数ですが，メディアに対し1バイトずつ（1文字ずつ）の操作をするものから1行単位の操作まで色々な形態を持っています。ここでは，これらについてサンプルプログラムを提示しますので，実行結果と解説を通して関数を理解して下さい。

(a) ファイル入出力の基本　【fopen，fclose，fcloseall】

ファイル入出力の基本は，何といってもこのfopenとfcloseです。関数名からでも想像出来るのですが，ファイルを開きファイルを閉じる関数です。ファイルというのは，普段はシッカリと鍵が掛けられていて，中身を見ることも取り出すことも入れることも出来ない箱のような物です。従って，この箱にデータを入れようとしたり，箱からデータを取り出そうとしたりするには，まず最初に箱を開けなければなりません。これがfopenです。また，このfopenには全くデータのない箱を注文する役目もあり，今まで使用していなかった新しい箱を作り出すことが出来ます。

これと反対に，色々な操作（データの読み込み，書き込み）が終わった箱をもと通りひっくり返しても大丈夫なように，ふたをして鍵を掛けなければなりません。これが，fcloseです。言い忘れましたが，この箱に付いている名前がファイルネームです。

【FOPEN】

書　式	fp＝fopen（fn，md）
引　数	str fn，md fnはファイルネームで，mdはopenするときのモードを指定します。fnでは，直接”（引用符）で囲んだファイルネームか，str型変数に代入されたファイルネームか，どちらかにより指定されます。 mdは，”r”，”w”，”rw”，”c”のいずれかで，次のような意味を持っています。 ”r”……既にファイルがあって，そのファイルを読み出し専用としてopenする場合。 ”w”……既にファイルがあって，そのファイルを書き込み専用としてopenする場合。 ”rw”…既にファイルがあって，そのファイルを読み書き両用に使用する場合。 ”c”……既にファイルがあっても，新規ファイルとして読み書き両用にopenする場合。この場合，既にあった貴重なデータは消えてしまうので，注意してopenしてください。
戻り値	int fp

	fp は，open により使用可能になったファイルの番号が入ります。この番号は，後で重要な役割をしますからきちんと戻り値として変数に代入しておいて下さい。

【FCLOSE】

書　式	fc＝fclose（fp）
引　数	int fp この fp は，open した時のファイル番号でどのファイルを close するかの指定になります。X-BASIC では，open により15個までのファイルを同時に開くことが出来ます。この中で，もう用のすんだファイルは close しますが，どのファイル番号かを指定しなければなりません。
戻り値	int fc これは，正常にファイルを close 出来たかどうかを戻り値として返します。正常な場合は０，異常（エラー）の場合は－１が返ります。

【FCLOSEALL】

書　式	fc＝fcloseall( )
引　数	ナシ
戻り値	int fc これは，先程の fclose と同様にファイルを閉じる関数ですが，引数がなく現在 open されているファイル全てが close されます。戻り値 fc に付いては fclose と同じです。

ファイル管理において，この二つ（fopen と fclose，fcloseall）は重要ですから，早いうちに理解して下さい。なお，fopen，fclose に付いての使用例は，次に説明する入出力関係の部分を参考にして下さい。

（b）１バイト単位の入出力　【fputc，fgetc】

この fputc,fgetc は，１バイトずつのデータを取り扱う関数です。１バイトデータというのは，半角のアルファベットや片仮名等の char 型のデータ一つで表すことの出来るデータのことです。１バイトずつ書き込みをするのが fputc で，１バイトずつ読み込むのが fgetc です。

【FPUTC】【FGETC】

書式	fc＝fputc (ch, fp), fs＝fgetc (fp)
引数	char ch (fgetcでは引数が一つだけ) int fp このchという引数は，これから書き込もうとするデータです。fpは，書き込もうとするファイルの番号で，fopenの戻り値です。
戻り値	● int fc この戻り値は，ファイルに対して正常に書き込めたかどうかを返すためのもので，正常な場合は0，異常の場合は－1を返します。 ● int fs……文字コード

さてこのfputc関数を使って，実際にファイルとしてディスクに書き込んでみます。まず，**第3－3図**を打ち込んでみて下さい。

```
 10 str fn="sample1.dat"
 20 dim char x(6)={'x','6','8','0','0','0',0}
 30 int fp,fs,i
 40 char t
 50 fp=fopen(fn,"c")
 60 for i=0 to 6
 70      t=x(i)
 80      fs=fputc(t,fp)
 90 next
100 fs=fclose(fp)
110 end
```

第3－3図　fputcサンプルプログラム

このプログラムでは，char型の配列変数に直接代入（初期化）した内容がフロッピーディスク内へ，sample1.datとして登録されます。新規の登録となるため，fopenでは"c"モードによりopenされています。

このプログラムを実行した後で，filesコマンドを実行してみると，今出来上がったばかりのファイルが確認出来ると思います。このファイルは，"X68000"という6文字と文字列の終わりを示すヌルコードから出来ているので，確認したファイルのサイズは7バイトになっていると思います。また，このプログラムの中でchar型の配列を初期化する時に，'(アポストロフィー)で囲んで

ありますが，これは，'で囲まれた文字のコードを意味しています（1文字だけの時に有効）。

それでは，今書き込んだファイルを読み込んでみましょう。**第3－4図**は，今書き込んだファイルをstr型の変数に代入して表示するプログラムです。

```
 10 str fn="sample1.dat"
 20 str s
 30 dim char x(6)
 40 int fp,fs,i,t
 50 fp=fopen(fn,"r")
 60 for i=0 to 6
 70     t=fgetc(fp)
 80     x(i)=t
 90 next
100 fs=fclose(fp)
110 for i=0 to 6
120     s=s+chr$(x(i))
130 next
140 print s
150 end
run
X68000
Ok
█
```

第3－4図　書き込んだファイルを表示させる（fgetc サンプルプログラム）

このように，先程のデータが取り出せた訳です。このプログラムの中で，chr$(　) というものがありますが，これは後述する**データ変換関数**で，「char型のデータをstr型のデータに変換する関数」です。このプログラムで注意しなければならないのは，第3－3図ではopenの時に"c"であったものを，第3－4図では"r"となっていることです。これは，既にあるファイルを読み出し専用にopenしたためで，このように使用目的によりモードを使い分けることが必要です。このプログラムの80行で少々無理のあることをしていますが，この代入のことを**型変換**といいます。型変換は，数値型の変数や値を互いに自由に代入することが出来ますが，オーバーフローを起こすとエラーになりますので気を付けて下さい。

（c）配列の入出力　【fwrite，fread】

これまで解説してきた入出力関数は，扱うデータが1バイト単位でした。ここで説明する関数は，扱う変数の型が基準となるものです。従って，書き込むデータがchar型だと1バイトずつ，int型であれば4バイトずつ，float型であれば8バイトずつ，一度に書き込まれていきます。ただしファイルの中へ入ってしまえば1バイトずつのデータですから，取り出す時に他の型へ取り込むこともできます。

【FWRITE】

書　式	fb＝fwrite（na，n，fp）
引　数	数値型1次元配列　na この引数は，ファイルへこれから書き込もうとするデータを格納しておく配列で，前述のように型により書き込まれるバイト数が変化します。 int n，fp この引数nは，書き込むデータの個数（カウント数）を指定するもので，1カウントはデータの型により変化します。char型の場合1バイトずつ，int型の場合4バイトずつ，float型の場合8バイトずつで1カウントになります。fpは，これから書き込もうとするファイル番号で，fopenの時の戻り値です。

それでは実際に，fwriteを使用したサンプルプログラムを実行してみます。この**第3－5図**のプログラムは，第3－3図のプログラムと同様に，"X68000"というデータを"sample2. dat"というファイルに書き込みます。

```
10 str fn="sample2.dat"
20 dim char x(6)={'X','6','8','0','0','0',0}
30 int fp,fb,fs,n=7
40 fp=fopen(fn,"c")
50 fb=fwrite(x,n,fp)
60 print fb;"バイト"
70 fs=fclose(fp)
80 end
run
7バイト
Ok
```

```
■
```

第３－５図　fwrite サンプルプログラム

このプログラムの場合，第３－３図と同じ動作にするため char 型の配列を使用しましたが，使用目的により数値型の別の配列にすることもできます。このプログラムに付いても，files コマンドにより出来上がったファイルを確認することが出来ます。

さて，出来上がったファイルを第３－４図の様に読み出すプログラムを作ってみます。今度は，fwrite とペアになる fread 関数を使います。

【FREAD】

書　式	fv＝fread（na，n，fp）
引　数	数値型１次元配列　na fwrite 同様，数値型１次元配列ですがこれは今までの関数とは少し違っていて，引数として渡した na にデータが入ってきます。 int n，fp これも fwrite 同様，読み込むデータの カウント数 n と，ファイル番号 fp です。
戻り値	int fv fv には実際に読み込んだカウント数が返ります。従って，char 型の場合１バイト，int 型の場合４バイト，float 型の場合８バイトで１カウントとなります。

この関数を使って，第３－４図同様，第３－５図で作った”sample2. dat”の内容を読み込んでみます（**第３－６図**）。

```
 10 str s,fn="sample2.dat"
 20 int fp,fv,fs,i,n=7
 30 dim char x(6)
 40 fp=fopen(fn,"r")
 50 fv=fread(x,n,fp)
 60 for i=0 to 6
 70      s=s+chr$(x(i))
 80 next
 90 print fv;"バイト"
100 print s
```

```
110 fs=fclose(fp)
120 end
run
7バイト
X68000
Ok
█
```

第3－6図　freadサンプルプログラム

（d）文字列の入出力　【fwrites，freads】

ここで説明するfwrites，freadsというファイル入出力関数は，今まで説明してきたファイル入出力関数（fputc，fgetc，fwrite，fread）を基本としてこれらの機能の拡張をした関数といえます。str型のデータは0から255までの文字コードの並びですから，char型の配列として扱うことが出来ます。char型の配列であれば，前述の配列のファイル入出力関数のfwrite，freadが使えます。さて，もう一つさかのぼってみます。配列のファイル入出力は，char，int，floatのそれぞれの型により扱われるバイト数が異なります。しかし，1，4，8バイトずつの操作であれば，全てchar型のデータとして処理が出来ます。となれば，fputc，fgetcに依ってデータの入出力が出来ます。

このようにfwrites，freads，は基本的にはfputc，fgetcの拡張関数になるわけですが，扱うstr型のデータは不定長です。というのは，変数のところでも説明しましたが，str型のデータは宣言する時に文字列の最大長さを設定することで，0から255文字までの範囲があります。そのため，この関数を使用する時にはこれまでになかった制限が付きます。この関数では，文字列一つを1行として扱います。従って，文字列の最後には1行の終了を示すコード（改行コード）を入れなければなりません。改行コードは文字コードの13と10で，16進表現で，&H0Dと&H0Aです。

【FWRITES】

書　式	fb＝fwrites（st，fp）
引　数	str st この引数は，これからファイルへ書き込もうとするデータ（文字列）を入れてあるもの。文字列の変数または文字列。 int fp この引数は，今書き込もうとしているファイルの番号を示すもので，fopenの時に戻り値として返ってきたもの。

戻り値	int fp 戻り値には，この関数により書き込まれたデータ（文字）の数が返ります。

では，実際に fwrites 関数を使っていくつかの文字列をファイルへ書き込んでみます。この**第3－7図**のプログラムでは，四つの str 型の変数を用意します。三つはそれぞれ x，y，z で X-BASIC，X68000，GOOD という文字列が初期化されています。

```
 10 str crlf:crlf=chr$(13)+chr$(10)
 20 str x="X-BASIC",y="X68000",z="GOOD"
 30 int fp,fb,fs,n
 40 fp=fopen("sample4.dat","c")
 50 fb=fwrites(x+crlf,fp)
 60 n=fb
 70 fb=fwrites(y+crlf,fp)
 80 n=n+fb
 90 fb=fwrites(z+crlf,fp)
100 n=n+fb
110 print n;"バイト"
120 fs=fclose(fp)
130 end
run
23バイト
Ok
■
```

第3－7図　fwrites サンプルプログラム

このプログラムの場合，直接扱っている文字数は x，y，z の変数に入っている文字列で，合計すると17バイトです。しかし fwrites を実行する時に，それぞれの文字列に crlf という文字列を付加しています。この crlf が先に説明した改行コードです。この改行コードは文字数として数えられますが，文字列の終わりを示す null コードは数えません。

【FREADS】

書　式	fb=freads（st，fp)

引　数	str st この引数は特殊なもので，これから読み出すデータが格納されてきます。従って，以前入っていたデータは失われますから注意が必要です。 int fp この引数は fwrites のものと同じです。

この関数を使用して，前に作成したファイルの中から**第３－８図**の様にデータを読み出してみます。第３－７図では，23バイトだった書き込み文字数は17バイトになっています。

```
 10 str x,y,z
 20 int fp,fb,fs,n
 30 fp=fopen("sample4.dat","r")
 40 fb=freads(x,fp)
 50 print fb
 60 fb=freads(y,fp)
 70 print fb
 80 fb=freads(z,fp)
 90 print fb
100 print x,y,z
110 fs=fclose(fp)
120 end
run
7
6
4
X-BASIC X68000  GOOD
Ok
■
```

第３－８図　freadsサンプルプログラム

ご覧のように，先程書き込んだデータがそれぞれの変数に読み込まれてきました。行の終わりを示す改行コードを入れるまでは１行として取り扱われますが，データが文字で長さが不定の場合には非常に有効な関数であることが理解してもらえたと思います。

(e) その他のファイル入出力関数 【fseek，feof，dskf】

これらの関数は，ファイルに対してこれから読み書きするデータの位置を知らせたり，記憶装置として，フロッピーディスクを使用した場合のフロッピーの空き容量を示したり，ファイルの終了情報を返したりするファイル入出力関数としては、補助的なものです。

【FSEEK】

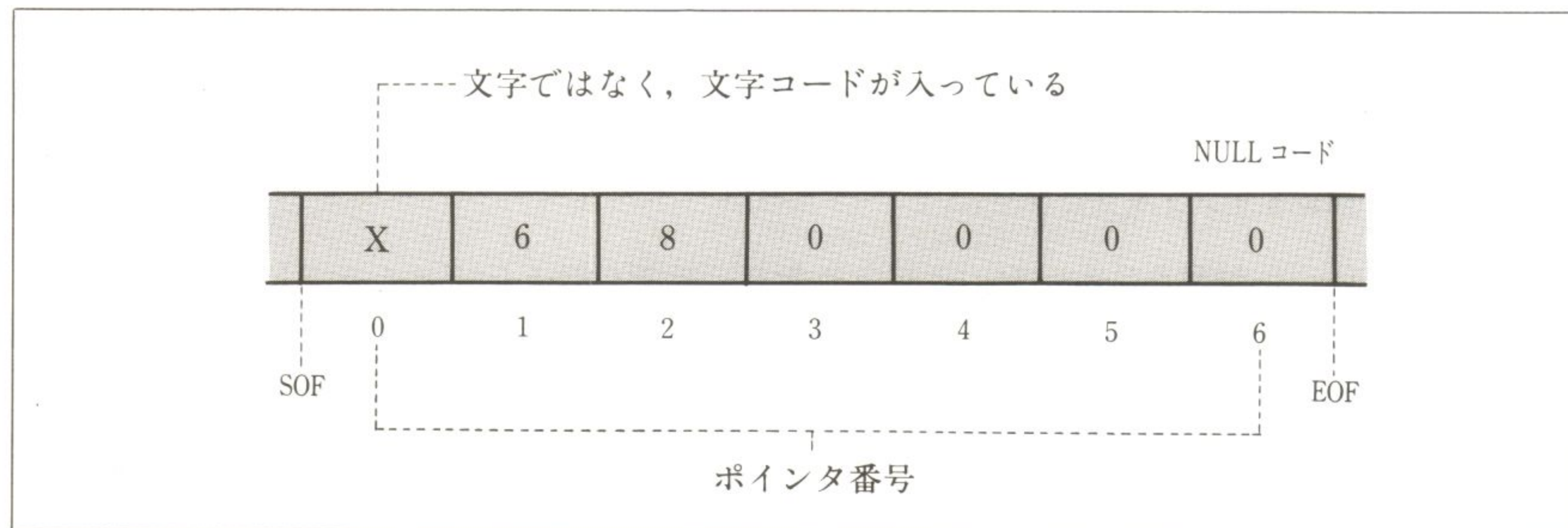
書　式	dp＝fseek（fp，os，md）
引　数	int fp，os，md この引数はファイルの指定をする fp，ファイルのシークする基準位置の指定をする md，基準位置からのオフセットを指定する os です。 ｏｓ……オフセットのバイト数。 ｍｄ……シークモードで次のような指定が出来ます。 ｍｄ＝0　　ファイルの初めから ｍｄ＝1　　現在のデータポインタから ｍｄ＝2　　ファイルの終わりから
戻り値	int dp この戻り値には，fseek 関数実行後のデータポインタの位置が返ります。この関数で，無効な引数（md でファイルの終わりを指定したにもかかわらず，オフセットをプラス側へ掛けたなど）の場合には，エラーとなり－1が返ります。

この fseek は，関数の書式，引数，戻り値の説明だけでは説明しきれない機能や使い方があります。サンプルプログラムを関数の機能を理解出来るように細かくして解説していきます。

まず，引数から解説します。fp のファイル番号に付いては，他の関数と同じです。md と os によりこの関数は決定されると言ってもよいほどで，重要なポイントになります。次にあげる**第3－9図**は fputc のところで作成した sample1．dat というファイルの中身です。

文字ではなく，文字コードが入っている
NULL コード

X	6	8	0	0	0	0
0	1	2	3	4	5	6

SOF　EOF
ポインタ番号

第3－9図　sample1.dat のファイルの内容

この中に，SOF と EOF というものがありますが，これはファイルの始めと終わりを意味しています。X68000という文字は，fputc では文字ではなく文字コードとして書き込みました。この状態で，sample1．dat いうファイルを open すると，データポインタは0の位置にあります。従って fgetc で実行した第3－4図のプログラムで X68000というデータが読み込めたわけです。しかし，プログラム作成時に，ファイルの途中から読み書きをしたいなどという要求は良くあることです。このような場合に，fseek が使われます。

では，実際に X68000の68000だけを取り出してみることにします。理屈としては，データポインタを X68000の6の所へ持っていけばよいわけですから，fseek の引数に，md＝0（ファイルの始めから）os＝1（データポインタを一つ進めた位置）と指定すればよいです。実際に**第3－10図**で確認して下さい。

```
 10 str fn="sample1.dat",s
 20 char d
 30 int fp,fs,i,t
 40 fp=fopen(fn,"r")
 50 dp=fseek(fp,1.0)
 60 for i=0 to 5
 70      t=fgetc(fp)
 80      d=t
 90      s=s+chr$(d)
100 next
110 fclose(fp)
120 print fs,s
130 end
run
1        68000
Ok
■
```

第3－10図　fseek サンプルプログラム

ご覧の通り，fseek によりデータポインタが一つ進んだことが確認出来，読み出した文字列も68000となります。fseek の引数の os の補足ですが，バイト単位であることから，fwrite や fread の時は使用している数値型配列の型により若干手を加えてやらなければなりません。fputc, fgetc では扱うデータが1バイト単位なので，オフセットもそのままの数値でしたが，int 型，float 型のデータを扱う場合にはそれぞれ4倍，8倍の数値を使用しなければなりません。

それでは、もう少しfseekを使って色々なことを実行させてみます。次にあげる**第3－12図**は，最初にあらかじめ30個の"＊"（アスタリスク）を書き込んであるファイル（sample4．dat）に任意の位置にキーボードから入力した文字を書き込むものです。まず最初にsample4．datを用意しなければならないので，**第3－11図**を入力，実行して下さい。次に，第3－12図を入力して下さい。

```
10 int fp
20 str sp="******************************"
30 fp=fopen("sample4.dat","c")
40 fwrites(sp+chr$(13)+chr$(10),fp)
50 close(fp)
60 end
```

第3－11図　sample4.datのファイルを作る

```
 10 int fp,p:str st,k
 20 fp=fopen("sample4.dat","rw")
 30 repeat
 40 input "position(0-29):",p
 50 input "character:",k
 60 print
 70 dp=fseek(fp,p,0)
 80 fs=fputc(asc(k),fp)
 90 until k="*"
100 fseek(fp,0,0)
110 freads(st,fp)
120 print st
130 fclose(fp)
140 end
```

第3－12図　fseekサンプルプログラム

このプログラムを実行すると，まずデータポインタを聞いてきますから，０から29までの数値を入力して下さい。次に，書き込みたい文字（キーから入力出来るもの）１文字入力してリターンキーを押すと，次のポジションを聞いてきます。これを続けていって，入力したい文字のとこ

ろに"＊"を入力すると30個の"＊"に先程指定した位置に入力された文字が入っている文字列が表示されます。もちろんファイルの中がそのように変更されているのです。このプログラムで中心となるのは，70行の fseek と100行の fseek です。70行の fseek では書き込みたい位置へデータポインタを移動して，100行の fseek ではファイルの中身全部を読み出すことから，ファイルのデータポインタをファイルの先頭に戻す作業をしています。

この他にも fseek には変わった使い方があります。しかしこれは，feof と深いかかわりがあるので次にまわします。

【FEOF】

書　式	fs＝feof（fp）
引　数	int fp 指定するファイル名で，fopen の時に戻り値として返されたもの。
戻り値	int fs 指定されたファイルが終了の位置にきているかが返ります。もし終了していればー1，まだ終了の位置ではない時は0が返ります。

この関数を使って，sample1. dat というファイルに何バイトのデータがあるかを調べてみます。sample1. dat は，fputc の説明のところで X68000という文字コードが入っていて，ヌルコードを含めて7文字（7バイト）のはずです。この**第3－13図**のプログラムは，sample1. dat を open した時にはデータポインタが0になっていることから，fgetc で1文字ずつ読み込んでいっては，feof でファイルの終わりかどうかをチェックします。

```
10 int fp,fs,n,t
20 fp=fopen("sample1.dat","r")
30 repeat
40 t=fgetc(fp)
50 n=n+1
60 until feof(fp)=-1
70 print "filesize=";n;"bytes"
80 fclose(fp)
90 end
run
filesize= 7bytes
Ok
```

```
■
```

第３－13図 feof サンプルプログラム

ここで，前述の fseek 関数の登場になります。ファイルのサイズを知るためには，fseek の md（シークモード）を２にしてＯＳ（オフセット）を０にして fseek を実行すれば，戻り値として現在のデータポインタ即ち，ファイルの一番最後の位置が返ってきます。この数値はこのファイルの大きさを意味しています。実際に第３－14図のプログラムで確認して下さい。

```
10 int fp,fs,n,t
20 fp=fopen("sample1.dat"."r")
30 n=fseek(fp,0,2)
40 print "filesize=";n;"bytes"
50 fclose(fp)
60 end
run
filesize= 7bytes
Ok
■
```

第３－14図　fseek でファイルサイズを読む

第３－13図と同じ結果が得られます。このように feof をファイルサイズをチェックするために使うのであれば，fseek の方が有効であるといえます。

【DSKF】

書　式	s=dskf（ｉ）
引　数	int i この引数は，これから調べたいフロッピーディスクを指定するもので，ドライブの指定をします。ドライブの指定は次のようになります。 ０……カレントドライブ １……Ａ　ドライブ ２……Ｂ　ドライブ
戻り値	int s 指定されたドライブのフロッピーディスクの空き容量を返します。

第3－15図のプログラムでは，現在ドライブAに入っているフロッピーの残り容量が表示されます。

```
10 print dskf(1)
20 end
```

第3－15図 dskf **サンプルプログラム**

これまでに，ファイル入出力関数に付いて解説してきましたが，X-BASICを使っていくうちに色々なデータをフロッピーに蓄えておきたくなったり，これから解説していく外部関数等で作った音楽演奏データや，グラフィックデータ等もフロッピーに記憶しておきたいなどの要求が出てきます。十分理解して，大切なデータを保管して下さい。使い方によっては便利な関数も，間違った使い方をすると大事なデータを消去させたりする凶器にもなりかねません。

3-1-2 データの変換関数

X-BASICには，char型，int型，float型，str型のデータなど，用途に応じた形のデータがありますが，プログラムを作っていくうちには必ずといってよいほど，それらを変換して違う形のデータにしたいということが起きてきます。例えば，10進法を16進法にしたいとか，数値のデータを文字列にしたいとかです。そういう時に便利なのが，データの変換の関数です。

一般的には，次のような形で表します。

戻り値＝関数名（引数）

これを，ASCという関数であてはめてみると

r＝asc（n）

nに引数の値を代入しておけば，rに結果が入ってくるというわけです。もちろんrとnは，あらかじめデータの型を宣言しておかなければなりません。ここで，引数はstr型で，戻り値はint型だとすると，**第3－16図**のようになります。

```
10  str n
20  int r
30  n = "A"
40  r = asc(n)
50  print r
60  end
```

```
run
65
Ok
■
```

第３－16図　データ変換関数の使用例

行番号10で引数を，行番号20で戻り値をそれぞれ宣言していますが，関数によってこの型は変わってきますので，それに合った型宣言が必要です。

さて，このデータ変換関数には，機能について分類してみると引数と戻り値に付いても分類出来ます。これをまとめると，**第３－17図**の様になります。

戻り値	**str型**・・・・・	BIN$,　OCT$,　HEX$ CHR$,　STR$,　GCVT ITOA,　ECVT,　FCVT
戻り値	**int型**・・・・・・	ASC,　ATOI,　INT TOASCII,　TOLOWER,　TOUPPER
戻り値	**float型**・・・・・	FIX,　ATOF,　VAL

第３－17図　データ変換関数

それでは，それぞれの関数に付いて説明します。

（ａ）戻り値がstr型の関数

戻り値がstr型の関数で，引数が数値型のものはそのものズバリ，数値を文字列に変換したい時に使用します。表現したい数値の形を変えるものもあり，またそのまま文字列にするものや色々とあります。次にあげる関数は，数値を文字列に変換する時にその数値の表現方法まで変えてしまうものです。

【BIN$】

int型のデータを２進数に変換して文字列として返します。上位の０は除かれます。

書　式	r＝bin$（n）
引　数	int n
戻り値	str r

この関数の動作を，次の**第 3 －18図**のサンプルプログラムで確認してみて下さい。

```
10 int x=15,y=16
20 str a,b
30 a=bin$(x)
40 b=bin$(y)
50 print a,b
60 end
run
1111      10000
Ok
■
```

第 3 －18図　BIN$サンプルプログラム

【OCT$】

int 型のデータを 8 進数の文字列に変換する関数です。

書　式	r =oct$(n)
引　数	int n.
戻り値	str r

この関数を次の**第 3 －19図**で確認して下さい。

```
10 int x=102
20 str a
30 a=oct$(x)
40 print a
50 end
run
146
Ok
```

```
■
```

第３－19図　OCT＄サンプルプログラム

【HEX＄】

int 型のデータを16進数の文字列に変換する関数です。上位に０がある場合は，取り除かれます。

書　式	r＝hex＄(n)
引　数	int n
戻り値	str r

この関数の確認は，次の**第３－20図**で行って下さい。

```
10 int x=128
20 str a
30 a=hex$(x)
40 print a
50 end
run
80
Ok
■
```

第３－20図　HEX＄サンプルプログラム

ここにあげた三つの関数は全て引数が int 型で，戻り値はその数値に対する２進表現，８進表現，16進表現です。このように数値の表現を変える関数もデータ変換になります。

次にあげる関数は，数値を文字にするものの中で考え方が特殊で，初めから文字であることを意識している数値（文字コード）を扱う時に使用します。前述のファイル入出力関数のところでも触れましたが，ファイルの中に入っているデータは文字であろうと数値であろうと全て数値として保存されています。このようなファイルの中のデータを文字として扱えるようにする関数が chr＄です。

【CHR＄】

ASC の反対で，キャラクターコードを引数にして文字を返す関数です。引数は，０～255の間で

なければなりません。

書　式	r＝chr＄(n)
引　数	char n
戻り値	str r

この関数を次の**第3－21図**で確認して下さい。この関数で扱われる引数は，char 型なので当然引数は0から255までのものとなりますが，色々な数値を代入して確認してみると，表示出来ないものもあります。このように表示出来ないものに付いては，X-BASICのマニュアルの巻末にコントロールコードとして紹介されています。コントロールコードに対応する機能を print 文により使用することが出来ます。

```
10 int x=65
20 str a
30 a=chr$(x)
40 print a
50 print a
60 end
run
A
Ok
■
```

第3－21図　CHR＄サンプルプログラム

次にあげる関数は，数値を文字列にそのまま，又は表示文字数に制限を付けたりその文字列の情報を分解して戻したりするものです。与えた引数をそのまま文字列に変換するといっても関数が存在している以上その必要性があってのことです。しかしどの関数でもいえることですが，使うのはユーザーですからいろんな使い方を見つけてみて下さい。

【ITOA】

int 型のデータを文字列に変換する関数です。

書　式	r＝itoa（n）

引　数	int n
戻り値	str r

この関数を，**第3－22図**により確認して下さい。この関数では，数値が文字列として扱われた場合の特長を確認することが出来ます。

```
10 int x=1000,y=10
20 str a,b
30 a=itoa(x)
40 b=itoa(y)
50 print x+y,a+b
60 end
run
1010 100010
```

第3－22図　ITOAサンプルプログラム

【STR$】

float 型のデータを、文字列に変換する関数です。

書　式	r =str$(n)
引　数	float n
戻り値	str r

この関数の確認は，次の**第3－23図**で行って下さい。

```
10 float x=10.1#,y=0.1#
20 str a,b
30 a=str$(x)
40 b=str$(y)
50 print a+b
60 end
run
```

```
10.10.1
Ok
■
```

第3－23図　STR$サンプルプログラム

【GCVT】

float型のデータを，指定された桁数の文字列に変換する関数です。2番目の引数によって桁数を指定します。

書　式	r＝gcvt（n1，n2）
引　数	1番目　float　n 1 2番目　int　　n 2
戻り値	str r

この関数で指定される桁数は，n 1の値によって桁数の意味が変わってきます。次の第3－24図でその違いを確認して下さい。

```
10 float x=0.111,y=12.11#
20 str a,b
30 a=gcvt(x,3):b=gcvt(y,3)
40 print a,b
50 end
run
0.111    12.1
Ok
■
```

第3－24図　GCVTサンプルプログラム

ご覧のように，どちらも指定桁数は3桁にしてありますが，結果は変数aが0.111，変数bが12.1となっています。これは小数点を含まず，小数点以上の数値がない場合は小数点以下の桁数が，小数点以上の桁がある時は，小数点以上の桁数を優先します。

これ以外に関数の結果が二つ以上の値を持つ特殊な関数があります。戻り値は，関数では1個

の値しか持てないので，それらは引数にも値を返しています。
　以下の関数はそういう特殊なものです。

【ECVT】

　float 型のデータを文字列に変換して戻り値とし，文字列の小数点以下の桁数と，小数点の位置と，プラス，マイナスの符号をそれぞれ int 型の引数に返します。

書　式	r＝ecvt（n1，n2，n3，n4）
引　数	n 1……float 型で与えます。 n 2……int 型で，文字列の小数点以下の桁数を返します。 n 3……int 型で，小数点の位置を返します。 n 4……int 型で，プラスのデータなら０を，マイナスなら１を返します。
戻り値	str r　数字だけの文字列が返ります。

【FCVT】

　float 型のデータを文字列に変換して戻り値とし，文字列の桁数と，小数点の位置と，プラス，マイナスの符号をそれぞれ int 型の引数に返します。

書　式	r＝fcvt（n1，n2，n3，n4）
引　数	n 1……float 型で与えます。 n 2……int 型で，文字列の桁数を返します。 n 3……int 型で，小数点の位置を返します。 n 4……int 型で，プラスのデータなら０を，マイナスなら１を返します。
戻り値	str r　数字だけの文字列が返ります。

（b）戻り値が int 型の関数

　これから紹介する関数は戻り値が int 型のものです。ということは，なにか（引数）を数値に変換することがわかります。このように，関数の引数と戻り値だけを評価するだけで，おおよその予想が立てられるのも関数の面白いところです。

【ASC】

　文字のキャラクターコード（ASCII コード）を返す関数です。引数２個以上の文字列の時は，

最初の文字のキャラクターコードを返します。

書　式	r＝asc（n）
引　数	str n
戻り値	int r

この関数はchr$の逆の変換をするものですが，戻り値はint型です。chr$では引数がchar型でしたから違うようですが，実際に戻り値として返されるのは文字コードなので機能的にはchr$の逆の変換ということになります。

戻り値としてのint型のデータはもちろん文字コードである以上，0から255までの数値です。従ってchar型の変数にこの戻り値を代入することが出来ます。このようにX-BASICでは，数値型データは型が違う変数同士で代入することが出来ます。これを型変換といいますが，桁あふれ（オーバーフロー）が起こるとエラーになってしまうので注意が必要です。この関数は次の**第3－25図**によって確認して下さい。

```
10 str x="X-BASIC"
20 int a
30 a=asc(x)
40 print a
50 end
run
88
Ok
■
```

第3－25図　ASCサンプルプログラム

【ATOI】

文字列をint型の数値に変換する関数です。文字列は，int型に変換できる内容でなければなりません。変換できない文字があると，そこで変換をやめます。また，文字列がnullの時にはエラーになります。

書　式	r＝atoi（n）

引　数	str n
戻り値	int r

この関数は先程の asc 関数の様に str 型のデータを int 型に変換するものですが，文字コードではなく文字列として表現されている数値を元の数値に変換するものです。この関数は itoa の逆の変換をするものです。この関数は**第3－26図**によって確認して下さい。

```
10 str x="0.001",y="53.5"
20 int a,b
30 a=atoi(x)
40 b=atoi(y)
50 print a+b
60 end
run
53
Ok
■
```

第3－26図　ATOIサンプルプログラム

これまでは，引数か戻り値のどちらかに str 型のものがありましたが，以下の関数には int 型と float 型と char 型しかありません。言い換えれば，数値のデータ変換の関数ということになります。しかし，文字を扱う時に使用する関数とも言えます。この中にある，toascii, tolower, toupper というものがそうです。

【INT】

float 型のデータの小数点以下を切り捨てて，int 型のデータとして返します。

書　式	r＝int（n）
引　数	float n
戻り値	int r

この関数は従来の BASIC でも使われていたものですが，変数の型やデータの型がはっきりしている X-BASIC の場合では，引数に float，戻り値に int となるのは言うまでもありません。こ

の関数は次の**第3－27図**で確認して下さい。

```
10 float x=12.345#
20 int a
30 a=int(x)
40 print a
50 end
run
12
Ok
■
```

第3－27図　INTサンプルプログラム

ご覧のように，引数の小数部分が切り捨てられてint型のデータに変換されました。

ここから紹介する三つの関数は，文字（アルファベット）の大文字を小文字に，小文字を大文字に，与えられた文字コードを0から127のアルファベットや記号の文字コードに変換するものです。しかし文字に変換するのではなく，文字コードに変換することに注意して下さい。

【TOASCII】

引数がASCII文字コードでなければ，ASCII文字コードに変換して，int型のデータとして返します。引数が，ASCIIの文字コードの場合には，そのままint型データとして返します。

書　式	r＝toascii（n）
引　数	char n
戻り値	int r

【TOLOWER】

引数が英大文字であれば英小文字に変換し，int型データとして返します。引数が英大文字でない時は，そのままint型データとして返します。

書　式	r＝tolower（n）
引　数	char n

戻り値	int r

【TOUPPER】

引数が英小文字であれば英大文字に変換し，int 型データとして返します。引数が英小文字でない時は，そのまま int 型データとして返します。

書　式	r=toupper（n）
引　数	char n
戻り値	int r

この関数を確認するプログラムは，次の**第3－28図**へまとめてありますから，違いや動作に付いて確認して下さい。

```
 10 char y='a',z='A'
 20 int s,t,u,v,w,x
 30 s=toascii(y)
 40 t=toascii(z)
 50 u=tolower(y)
 60 v=tolower(z)
 70 w=toupper(y)
 80 x=toupper(z)
 90 print s,t,u,v,w,x
100 end
run
97        65        97        97        65        65
```

第3－28図　TOASCII，TOLOWER，TOUPPER サンプルプログラム

それぞれの引数が関数の機能どおりに変換されています。この第3－28図の中に　'(アポストロフィー）で囲まれたアルファベットがありますが，これはその文字が示す文字コードを意味するもので，char 型の変数へ数値で代入するのではなく，直接文字を表記してその文字コードを代入しようとするものです。こうして考えてみると，このアポストロフィーで囲まれた表現は，引数が1文字の str 型のデータを char 型の文字コードに変換する関数であると言えます。

（c）戻り値がfloatの関数

【ATOF】

文字列をfloat型の数値に変換する関数です。文字列は，float型にできる内容でなければなりません。変換できない文字があると，そこで変換をやめます。また，文字列がnullの時は，エラーとなります。小数点や指数を示すeまたはEがある時は，その前後に1個ずつ以上の数字がなければいけません。

書　式	r＝atof（n）
引　数	str n
戻り値	float r

この関数はstr＄の逆の変換をする関数です。しかし，数値を文字列にすることは何の制約もありませんが，文字列を数値に変換する時には色々と注意しなければならないことがあります。それは，数値といえば0から9までの数字と小数点の．（ピリオド）や指数を表すeまたはEと符号の＋や－があるぐらいですが，文字列といってしまえば必ずしも上記の文字で構成されている保証はありません。これは文字列を数値に変換する時の重要なポイントです。この関数は**第3－29図**で確認して下さい。

```
10 str x="0.001".y=0.1t5"
20 float a,b
30 a=atof(x)
40 b=atof(y)
50 print a,b
60 end
run
0.001    0.1
Ok
■
```

第3－29図　ATOFサンプルプログラム

【FIX】

float型のデータの小数点以下を切り捨てて，float型のデータとして返す関数です。

書　式	r=fix（n）
引　数	float n
戻り値	float r

この関数はfloatをintに変換するint関数と同じ働きをしますが，戻り値はfloat型のままです。動作はint関数と同じです。

【VAL】

文字列データをfloat型のデータに変換する関数です。atof関数と同じ働きをします。

書　式	r=val（n）
引　数	str n
戻り値	float r

3-1-3　文字列処理

（a）文字列の種類をチェックする関数

文字列処理関数のなかで，IS××××という関数は，1文字のデータに対してそれがどんな種類の文字であるかをチェックする関数です。1文字とはいっても，そのチェックの対象となるのは，char型の数値です。変数の説明にもありましたが，char型は0～255の数値を扱うデータ型ですが，その用途は文字データの取扱いに適しています。ここで説明するIS××××という関数は，全てこのchar型のデータに対して，それがチェックの対象となるかどうか調べ，int型のデータの0か－1を返すものです。ですから，引数は必ずchar型で，戻り値はint型になります。従って，実際にこの関数を使用する場合，文字列の中のどの文字を調べたいか，またその1文字だけをどのように取り出せばよいのかを十分に知っている必要があります。

ここで少しその方法について，簡単なものを説明しておきましょう。

第3－30図のプログラムのように変数aをstr型として宣言し，”X-BASIC”という文字列が入っているとします。その中の3番目の文字を取り出したい時，a［2］と書くとchar型として3

番目の'B'のデータ（数値としては,66)が取り出せるのです。a［2］の2は，0から数えて3番目が2になるからです。1番目の時はa［0］と書きます。こういう方法でchar型データが取り出せるのは，おそらくX-BASICの内部でstr型という型が，char型の配列と同じような扱いになっているからだと思われますが，とにかく覚えておくと便利なことでしょう。

```
10  str a = "X-BASIC"
20  print a[2]
run
66
Ok
■
```

第3－30図　文字列より任意のデータを取り出す

これから説明する関数は全て引数がchar型で，戻り値がint型です。また，それぞれの関数の違いは何を調査対象にするかで、調べ方及び調べた結果の表現のしかたは全く同じですから，基本的な使用例を**第3－31図**に示しておきます。

```
10 char n='□
20 int r
30 r=isxxxxx(n)
40 print r
50 end
run
0 または -1
Ok
■
```

第3－31図　IS××××関数の基本的な使用例

【ISALNUM】

データが英数字であるかどうか調べ，英数字であれば－1を、そうでなければ0を返します。英数字とは英大文字（A～Z)，英小文字（a～z)，数字（0～9）の文字コードです。

【ISALPHA】

データが英字であるかどうか調べ，英字であれば－1を，そうでなければ0を返します。英字とは英大文字（A～Z)，英小文字（a～z）の文字コードです。

【ISASCII】

文字コードのデータがアスキー文字であるかどうか調べ，アスキー文字であれば－1を，そうでなければ0を返します。アスキー文字とは，文字コードが0～127（&H00～&H7F）の文字です。

【ISCNTRL】

文字コードのデータがコントロール文字かどうか調べ，コントロール文字であれば－1を，そうでなければ0を返します。コントロール文字とは，文字コードが0～31と127（&H00～&H1Fと&H7F）の文字です。

【ISDIGIT】

文字コードのデータが数字であるかどうか調べ，数字であれば－1を，そうでなければ0を返します。数字とは，0～9のことです。

【ISGRAPH】

文字コードのデータがスペース以外の表示可能な文字かどうか調べ，そうであれば－1を，そうでなければ0を返します。スペース以外に表示可能な文字とは，文字コードが33～126（&H21～&H7E）の文字です。

【ISLOWER】

文字コードのデータが英小文字であるかどうか調べ，英小文字であれば－1を，そうでなければ0を返します。英小文字とは，a～zの文字です。

【ISPRINT】

文字コードのデータが表示可能な文字かどうか調べ，そうであれば－1を，そうでなければ0を返します。表示可能な文字とは，文字コードが33～126（&H21～&H7E）の文字で，スペースも含んでいます。

【ISPUNCT】

文字コードのデータが表示可能な記号かどうか調べ，そうであれば－1を，そうでなければ0を返します。表示可能な記号文字とは，文字コードが32～47，58～64，91～96，112～126（&H20～&H2F，&H3A～&H40，&H5B～&H60，&H7B～&H7E）の文字です。

【ISSPACE】

文字コードのデータが空白文字かどうか調べ，そうであれば－1を，そうでなければ0を返します。空白文字とは，文字コードが9～13，32（&H09～&H0D，&H20）の文字です。

【ISUPPER】

文字コードのデータが英大文字であるかどうか調べ，英大文字であれば－1を，そうでなければ0を返します。英大文字とは，A～Zの文字です。

【ISXDIGIT】

文字コードのデータが16進の文字であるかどうか調べ，そうであれば－1を，そうでなければ0を返します。16進の文字とは，0～9，A～F，a～fの文字です。

(b) 文字検索をする関数

ここに出てくる関数は，文字列中から指定されたもの（文字　または　文字列）を探し出しその位置を返すものです。

【INSTR】

文字列の中から文字列を探して，何文字目にあるかをint型で返す関数です。何文字目以降から探すという指定が出来ます（1～255の間)。どちらかの文字列がnullだったり，見つからなかった時や，探しはじめる位置が0以下だと，戻り値は0になります。

書　式	r＝instr（n 1，n 2，n 3）
引　数	n 1……int型で，何文字目から探すかの指定 n 2……str型で，この文字列中から探す n 3……str型で，探したい目的の文字列
戻り値	int r 探し出した文字の位置を返します
使用例	10　str ast="super central processing unit 68000" 20　int r 30　r=instr(1,ast,"unit") 40　print r 50　r=instr(10,ast,"ce") 60　print r run 26 18

	0k ■

【STRCHR】

文字列の中から1文字を探し，最初に見つけた位置を返します。先頭を0としたint型の値を返しますが，見つからなかった時は－1を返します。

書　式	r＝strchr（n 1，n 2）
引　数	n 1……str型で，この文字列から探します。 n 2……char型で，探したい文字を与えます。
戻り値	int r
使用例	10　str st="WE need real time multi task" 20　char a='t' 30　int r 40　r=strchr(st,a) 50　print r run 13 0k ■

【STRCSPN】

文字列1の中から，文字列2のどれか1文字を最初に見つけた位置を返します。戻り値は先頭を0としたint型の値を返します。見つからなかった時は，文字列の長さを返します。

書　式	r＝strcspn（n 1，n 2）
引　数	n 1……str型 n 2……str型で，探したい文字を並べます。

戻り値	int r
使用例	10 str a="super personal work station" 20 str b="abc" 30 int r 40 r=strcspn(a,b) 50 print r run 12 Ok ■

【STRSPN】

文字列の中から，指定した別の文字列にない文字を最初に見つけた位置を返します。見つからなかった時は，最初の文字列の長さを返します。また，どちらかの文字列がnull文字の時は，0を返します。

書　式	r＝strspn（n１，n２）
引　数	n１……str型で，この文字列の中から探します。 n２……str型で，この文字列の中の文字を比較します。
戻り値	int型

【STRTOK】

文字列の中から別に指定する文字列のうち，どれかひとつを見つけた位置までの文字列に返します。

書　式	r＝strtok（n１，n２）
引　数	n１……str型で，この文字列の中を探します。 n２……str型で，探したい文字を並べます。
戻り値	str型

【STRCHR】

文字列の中で指定した1文字が、最初に見つかった位置を返します。先頭は0で，最後まで見つからなかった時は，－1を返します。

書　式	r＝strchr（n1，n2）
引　数	str　　n1 char　　n2　キャラクタコードで変換
戻り値	int r

（c）　文字列を加工する関数

ここで紹介する関数は，ある条件によって文字列を切ったり貼ったりするものです。文字列処理のメインですから良く理解して下さい。

【LEFT＄，MID＄，RIGHT＄】

この三つの関数は，いずれも文字列の中から指定された一部分の文字列を取り出すためのものです。LEFT＄は左側から，RIGHT＄は右側から指定された数の文字列を取り出します。MID＄だけ，引数がひとつ多くて，何文字目から何個取り出すという指定ができるので，文字列の中央あたりから取り出すことが可能です。

書　式	r＝left＄(n1，n2) r＝right＄(n1，n2) r＝mid＄(n1，n2，n3)
引　数	1……str 型 2……int 型　LEFT＄は何文字取り出すかという指定 int 型　RIGHT＄は何文字取り出すかという指定 int 型　MID＄は何文字目からという指定 3……int 型　MID＄のときだけ必要。何文字取り出すかの指定
使用例	10　str st="friendly OS Human 68k" 20　str r 30　r=left$(st,6) 40　print r

	50　r=right$(st,9) 60　print r 70　r=mid$(st,10,2) 80　print r run friend Human 68k OS Ok ■

【MIRROR$】

文字列の並びを反転して返す関係です。

書　式	r =mirror$(n)
引　数	str n 関数実行後もnはそのまま
戻り値	str r 引数の文字列nが反転されて返されます
使用例	10　str st="X-BASIC" 20　str ts 30　ts=mirror$(st) 40　print ts run CISAB-X Ok ■

【STRING$】

一つの文字を指定した数だけ並べた文字列を作ります。長さは，最大255までで，0以下を指定

した時はnull文字を返します。

書　式	r＝string＄（n１，n２）
引　数	n１……int型で作る文字列の長さ n２……str型で１文字を指定
戻り値	str型
使用例	10　int a=5 20　str b="@" 30　str r 40　r=string$(a,b) 50　print r run @@@@@ Ok ■

【STRLEN，（LEN）】

文字列の長さを返します。null文字の時は0を返します。

書　式	r＝strlen（n） r＝len（n） （どちらでもよい）
引　数	str n
戻り値	int r

【STRLWR】

文字変数の中の英大文字を，英小文字に変換します。

書　式	r＝strlwr（n）

引　数	str n
戻り値	str r

【STRNSET】

文字変数の中の先頭から指定した数の文字を，指定した文字に変換します。

書　式	r＝strnset（n 1，n 2，n 3）
引　数	n 1……str 型変数 n 2……char 型　キャラクタコードで指定 n 3……int 型　何文字変換するかを指定

【STRREV】

文字変数の並びを逆転します。

書　式	r＝strrev（n）
引　数	str n 引数として渡した文字列nも反転するので注意して下さい。
戻り値	str r nが反転して返されます。

【STRSET】

文字変数の中の文字を，全て指定したキャラクタコードの文字にします。

書　式	r＝strset（n 1，n 2）
引　数	n 1……str 型変数 n 2……char 型　キャラクタコードで指定
戻り値	str 型

【STRUPR】

文字変数の中の英小文字を，全て英大文字に変換します。

書　式	r＝strupr（n）
引　数	str n
戻り値	str r

3-1-4 数値演算関数

ここに出てくる関数は，これまでに出てきたものより馴染みの深いものです。数学的な関数の原点のようなものですから理解しやすいと思います。幾つかの関数がありますが，数学的な分類や使用目的により分類してあります。

（a）三角関数

【ATAN】

nの逆正接（アークタンジェント）を計算します。戻り値としては，$-\pi/2$から$\pi/2$までの値を返します。

書　式	r＝atan（n）
引　数	float n
戻り値	float r

【SIN】

n（ラジアン）の正弦（サイン）を計算します。

書　式	r＝sin（n）
引　数	float n（ラジアン）
戻り値	float r

【COS】

n（ラジアン）の余弦（コサイン）を計算します。

書　式	r＝cos（n）
引　数	float n （ラジアン）
戻り値	float r

【TAN】

n（ラジアン）の正接（タンジェント）を計算します。

書　式	r＝tan（n）
引　数	float n （ラジアン）
戻り値	float r

（b）　対数演算関数

【EXP】

自然対数の底”e”のn乗を計算します。

書　式	r＝exp（n）
引　数	float n
戻り値	float r

【LOG】

nの自然対数を返します。n＞0でなければなりません。

書　式	r＝log（n）
引　数	float n （n＞0）

戻り値	float r

(c) 乱数用の関数

【SRAND】

rand（　）の系列を初期化します。iは，0～65535の範囲を取ります。

書　式	r＝srand（i）
引　数	int
戻り値	void　（戻り値はない）

【RAND】

0以上32767以内の整数型乱数を返します。

書　式	r＝rand（　）
引　数	なし
戻り値	int r　（0 ＜r ＜ 32767）

【RANDOMIZE】

rnd（　）の乱数系列を初期化します。

書　式	randomize（i）
引　数	int i i……乱数のシード（－32768≦ i ≦32767）
戻り値	void　（戻り値はない）

【RND】

0以上1未満の乱数を返します。

書　式	r＝rnd（　）
引　数	なし
戻り値	float　r（0≦r≦1）

（d）　その他の数値演算関数

【PI】

円周率を返します。

書　式	r＝pi（n）
引　数	float n n……倍数
戻り値	float r πがn倍になり返されます

【ABS】

nの絶対値を返します。

書　式	r＝abs（n）
引　数	float n
戻り値	float r

【POW】

xのy乗の計算をします。x＝0かつy＜0の場合と，x＜0かつyが整数でない場合は，エラーとなります。

書　式	r ＝pow（x，y）
引　数	float x，y　ただしx＜0で，yは整数値
戻り値	float r r ＝x^yの計算結果が返る

【SGN】

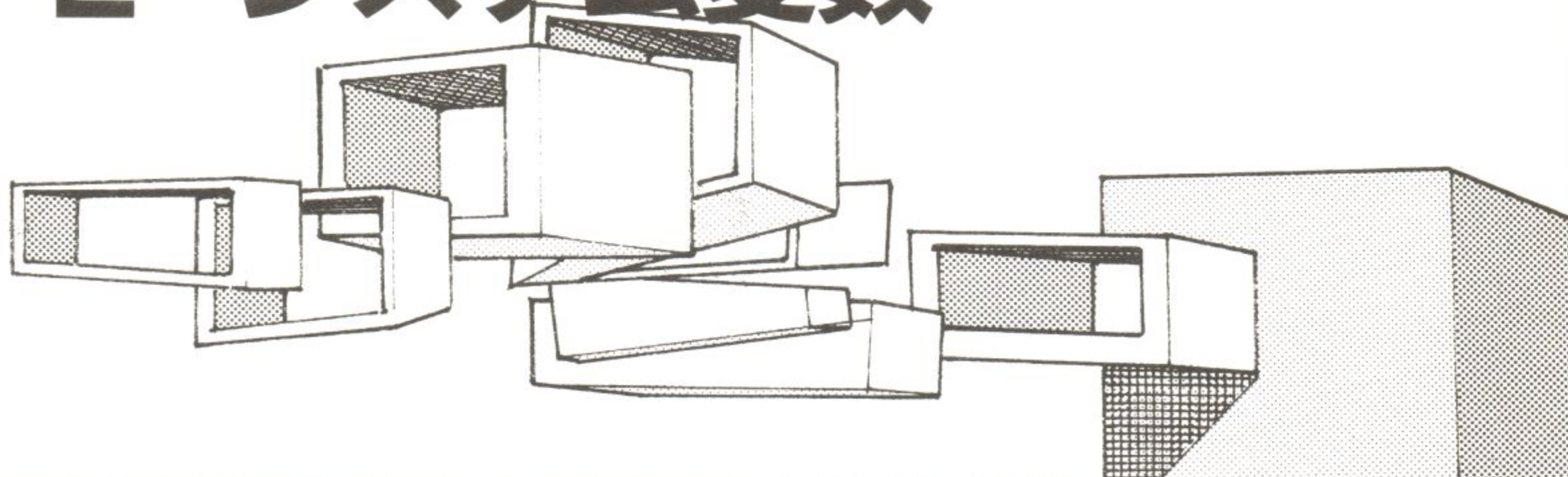
nの符号を調べ正の数だったら１を，１の場合は０を，負の数だったら－１を返します。

書　式	r ＝sgn（n）
引　数	float n
戻り値	float r

【SQR】

nの平方根（ルート）を返します。n≧０でなければなりません。

書　式	r ＝sqr（n）
引　数	float n
戻り値	float r

3-2 システム変数

ここでは標準関数にもステートメントにもなれなかった特殊な変数に付いて解説します。この特殊な変数のことを，**システム変数**といいます。このシステム変数は，X68000のメモリ上で色々なハードウエアの状態を各部が処理をするごとに変更される変数で，それぞれの情報が入るメモリに変数名を割り付けたものです。

システム変数で読み取れるハードウエアの状態とは次のものをいいます。

(a) プログラムメモリの残量
(b) 現在の日時・曜日
(c) キーボードからの情報
(d) カーソルの位置

これらは，X68000の各部で使用されているLSIの状態がメモリに書き込まれていると理解すればよいのです。

(a) プログラムメモリの残量を知らせる変数

プログラムメモリの大きさは，"BASIC.CNF" により設定され X-BASIC が立ち上がる時に，その大きさだけのメモリを確保します。このプログラムメモリは，プログラムモードでプログラムを打ち込んでいったり，実行中に大きな配列等を確保したりすることでプログラムメモリの残量が小さくなっていきます。そんな時にこの変数を読むことで，残りのメモリの量を知ることが出来ます。

【FREE】

書　式	n＝free または free
使用例	10 print free 20 end run 128030…………ユーザーの使用状況やメモリ設定により異なる

	0k ■

この変数は，読み込み専用で変数に値を代入することは出来ません。

(b) 現在の日時・曜日を知らせる変数

X68000には，バッテリーバックアップ付の時計（電源を切っても時を刻んでいる）が内蔵されています。このデータをいつでも新しい日時，曜日データに書き替えているのが次の三つのシステム変数です。

【DATE$】

書　式	st=date$　または　date$
使用例	10 print date$ 20 end run 87/11/07 0k ■

【DAY$】

書　式	st=day$　または　day$
使用例	10 print day$ 20 end run 土 0k ■

【TIME$】

書　式	st=time$　または　time$
使用例	10 print time$

	20 end run 14:23:16 Ok ■

なお，この変数は，全て str 型の変数です。この中で，date＄，time＄に付いては文字列を代入することが出来ます。代入される文字列の書式は，変数を出力してくるものと同じにしてなければなりません。この機能を利用して，時刻や日付をセットすることが出来ます。この機能の中に day＄をセットすることが出来ないのは，日付が決まってしまえば曜日も決まってしまうためです。

（c）キーボードからの情報を知らせる変数

ステートメントにあった input 文と似ていますが，str型の変数に1文字だけ入力されます。またこのシステム変数は，代入形式をとっているためこのシステム変数にはもう一つの str 型変数を用意しなければなりません。

【INKEY＄】

書　式	st＝inkey＄
使用例	10 str st 20 a=inkey$ 30 print st 40 end

この変数の代入文を見つけると，プログラムは中断しこの変数の中にキーボードからの文字データが入ってくるまで待ちます。キーボードが押されキーコード情報（文字コード）が左辺に代入されたところで，プログラムは再開されます。つまり，上記の例では，変数 a に代入されたところからプログラムが再開されます。このシステム変数では，1 文字入力の input 文（リターンを押す必要がない）として使用出来ます。このシステム変数でマニュアルに記述されていないものがあります。次の inkey＄（0）です。

【INKEY＄（0）】

書　式	st＝inkey＄（0）
使用例	10 str st

```
20 while 1
30      st=inkey$(0)
40      print st
50 endwhile
60 end
```

このプログラムが実行されると，キーを押した文字が表示されて，キーを離すと表示されなくなります。つまり，30行と40行を高速でループしてキーが押されていなくても次の処理へ移ります。このシステム変数を使えば，リアルタイムのキー入力が出来ます。例えば，表示中にキー入力により表示を一時ストップさせたり，表示をキャンセルさせたりすることが出来ます。

（d）カーソルの位置を知らせる変数

ステートメントのlocate文を使ってカーソル位置を指定することが出来ましたが，このシステム変数は現在どの位置にカーソルがあるのかを知るための変数です。この変数は読み出し専用で，数値を代入してカーソルを移動することは出来ません。カーソルを移動するにはlocate文を使用すればよいわけです。

【CSRLIN，POS】

書　式	y＝csrlin　または　csrlin x＝pos　　　　　　　pos
使用例	10 locate 10,15 20 print "x=";pos 30 print "y=";csrlin 40 end

以上がシステム変数ですが，X-BASICではこのシステム変数と同一名の変数を使用することが出来ません。また，この他にもステートメントや関数名と同一の変数名を使用することも出来ません。これらを，予約語と言ってX-BASICマニュアルの巻末に列記してありますので，そのまま使用しないで下さい。これらは，全く同じでなければ使用することが可能で，次のような工夫をすればプログラム中に変数として使用することが出来ます。

予約語	変数として使用可能になったもの
ABS	ABS_A
BOX	BOXM

NEXT FSEEK WHILE	NXT ── Eを削除 FSEEKS WHIL

ワンポイントテクニック　論理演算子

論理演算というのは複数の条件式や，ビット単位の処理をしたりする時に使用します。X-BASICでは，次のような論理演算の使用が許されています。

①否定　NOT

X	NOT X
0	1
1	0

②論理積　AND

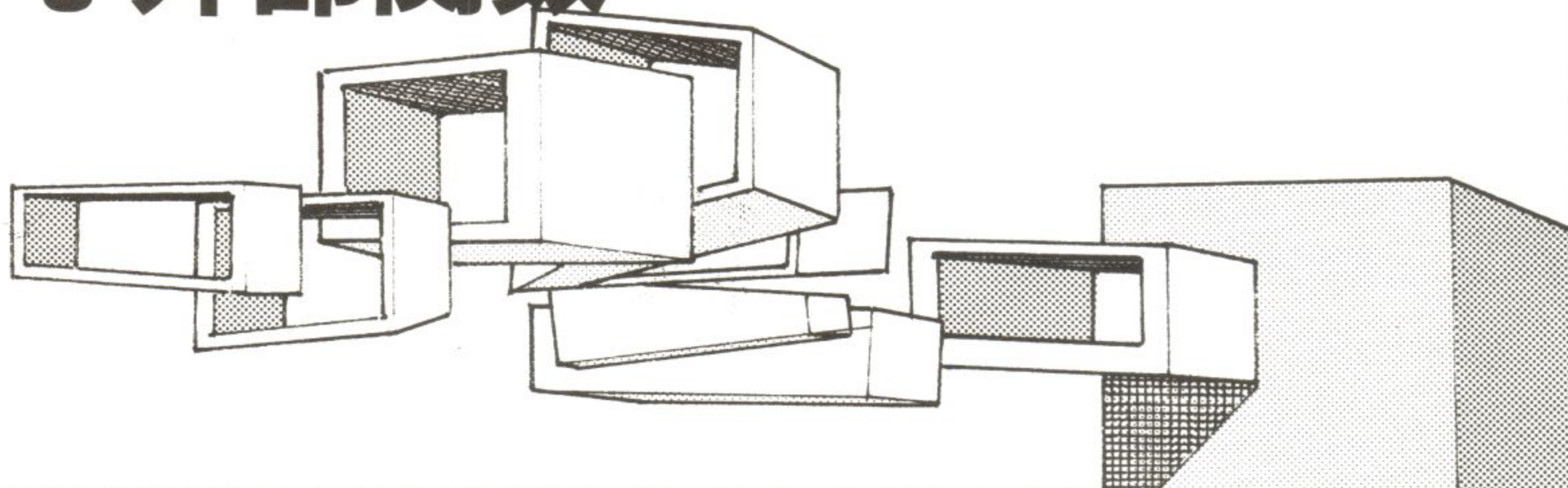
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

③論理和　OR

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

④排他的論理和　XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

3－3 外部関数

3-3-1 グラフィック関数

（１）画面モード

グラフィック表示のためには，まずグラフィック画面のモードを，使い方に合わせて設定する必要があります。このグラフィック画面のモードを設定するステートメントが，「SCREEN」です。

【SCREEN】

書　式	screen a，b，c，d
	a……表示画面サイズ a＝０のとき　256×256 a＝１のとき　512×512 a＝２のとき　768×512
	b……実画面，色モード及び使用可能なグラフィックページ b＝０のとき　1024×1024　65536色中16色　ページ０のみ b＝１のとき　512×512　65536色中16色　ページ０～３の４ページ b＝２のとき　512×512　65536色中256色　ページ０～１の２ページ b＝３のとき　512×512　65536色　ページ０のみ
	c……ディスプレイの解像度 c＝０のとき　標準解像度（15kHz）スーパーインポーズ可能 c＝１のとき　高解像度（31kHz）　スーパーインポーズ不可

	d……グラフィック画面の表示　ON/OFF d＝0のとき　グラフィックは表示されません。 d＝1のとき　グラフィック画面を初期化し表示ONにします。 このとき，グラフィックRAM内のデータは全て消去されます。

●グラフィックページ

グラフィック画面にはページという考え方があり，例えば，実画面512×512色モードが65536色中16色とした場合は，ページ0からページ3までの四つのページが実画面512×512，65536色中16色というモードで使えることになります。複数のページを扱う場合には，何ページ目を表示ON，またはOFFにするか，何ページ目に四角形を書くかといったように処理によってページを選択する必要があります。このページを扱うための関数が，vpageとapageです。

【VPAGE】

書　式	vpage（i）
引　数	int i

○……表示ON
×……表示OFF

iの値	ページ0	ページ1	ページ2	ページ3
0	×	×	×	×
1	○	×	×	×
2	×	○	×	×
3	○	○	×	×
4	×	×	○	×
5	○	×	○	×
6	×	○	○	×
7	○	○	○	×
8	×	×	×	○
9	○	×	×	○
10	×	○	×	○
11	○	○	×	○
12	×	×	○	○
13	○	×	○	○
14	×	○	○	○
15	○	○	○	○

iの値は，「SCREEN」で設定した画面モードによって決まります。

実画面サイズ	1024×1024（16色モード）	i＝0，1
	512×512（16色モード）	i＝0～15
	512×512（256色モード）	i＝0～3
	512×512（65536色モード）	i＝0，1

となります。

【APAGE】

書　式	apage（pag）
引　数	int pag pag……アクティブページ番号

アクティブページというのは，実際に読み書きするページのことを意味します。

第3－32図は画面モードを表示画面サイズ512×512，実画面サイズ512×512，65536色中16色モードで，０～３ページのグラフィックページを使用できるように設定し，各ページに違う色の四角形を書き vpage 関数の i の値をかえることで，どのように四角形が表示されているかを実験してもらうためのプログラムです。実行させると，vpage（0）の状態からはじまり，何かキーを押すごとに vpage 関数の i の値が変化していきます。

```
10 /* apage&vpage test
20 int i
30 str space
40 screen 1,1,1,1
50 palet(1,hsv(0.31,31))
60 palet(2,hsv(32,31,31))
70 palet(3,hsv(64,31,31))
80 palet(4,hsv(96,31,31))
90 vpage(0)
100 apage(0)
110 fill(0,0,60,60,1)
120 apage(1)
```

```
130 fill(61,61,120,120,2)
140 apage(2)
150 fill(121,121,180,180,3)
160 apage(3)
170 fill(181,181,240,240,4)
180   for i=0 to 15
190       locate 30,2:print"vpage(";i;")"
200       vpage(i)
210       space=inkey$
220   next
230 end
```

第３－32図　apage, vpage サンプルプログラム

10：コメント文
20～30：変数宣言
40：画面モードを512×512の16色モードに設定
50～80：パレットコード１～４に赤，黄，緑，シアンの順にセット
90：グラフィックページ０～３をすべて表示OFFにする。
100：アクティブページをページ０に設定
110：ページ０に赤の四角形を描く。
120：アクティブページをページ１に設定
130：ページ１に黄の四角形を描く。
140：アクティブページをページ２に設定
150：ページ２に緑の四角形を描く。
160：アクティブページをページ３に設定
170：ページ３にシアン（水色）の四角形を描く。
180：220行の next 間を16回(０～15)ループする。この時，ｉの値は１つずつ増加する。
190：ｉの値を画面に vpage（ｉ）の形式で表示する。
200：vpage 関数で各グラフィックページを ON，OFF させる。この値は０～15まで１つずつ増加する。
210：何かキーが押されるまで待つ。
220：next 文

●クリッピング

クリッピングというのは，表示範囲よりはみ出した部分を取り去ることをいいます。この表示範囲というのは，window 関数で指定された範囲となります。

【WINDOW】

書　式	window（x1，y1，x2，y2）
引　数	int x1，y1，x2，y2 x1……表示指定範囲の左ｘ座標 y1……表示指定範囲の左ｙ座標 x2……表示指定範囲の右ｘ座標 y2……表示指定範囲の右ｙ座標

・各引数は画面モードや設定された実画面サイズにより，0～511あるいは0～1023の値となります。

・screenを実行すると，その表示画面サイズが表示範囲となります。

第３－33図は画面の中心から半径を５ずつ増加させて円を書くプログラムですが，表示範囲をwindow関数で変えていますので，クリッピングというものがどのようなものか，実際に確かめてみて下さい。

```
 10 /* window test
 20 screen 2,0,1,1
 30 palet(1,hsv(32,31,31))
 40 palet(2,hsv(96,31,31))
 50 window(100,100,300,300)
 60 print " window(100,100,300,300)"
 70 box(100,100,300,300,2)
 80 en()
 90 window(300,200,500,500)
100 print " window(300,200,500,500)"
110 box(300,200,500,500,2)
120 en()
130 window(0,0,767,511)
140 print " window(0,0,767,511)"
150 box(0,0,767,511,2)
160 en()
170 end
180 func en()
```

```
190      int i
200      for i=1 to 50
210         circle(384,256,i*5,1,0,360,256)
220      next
230 endfunc
```

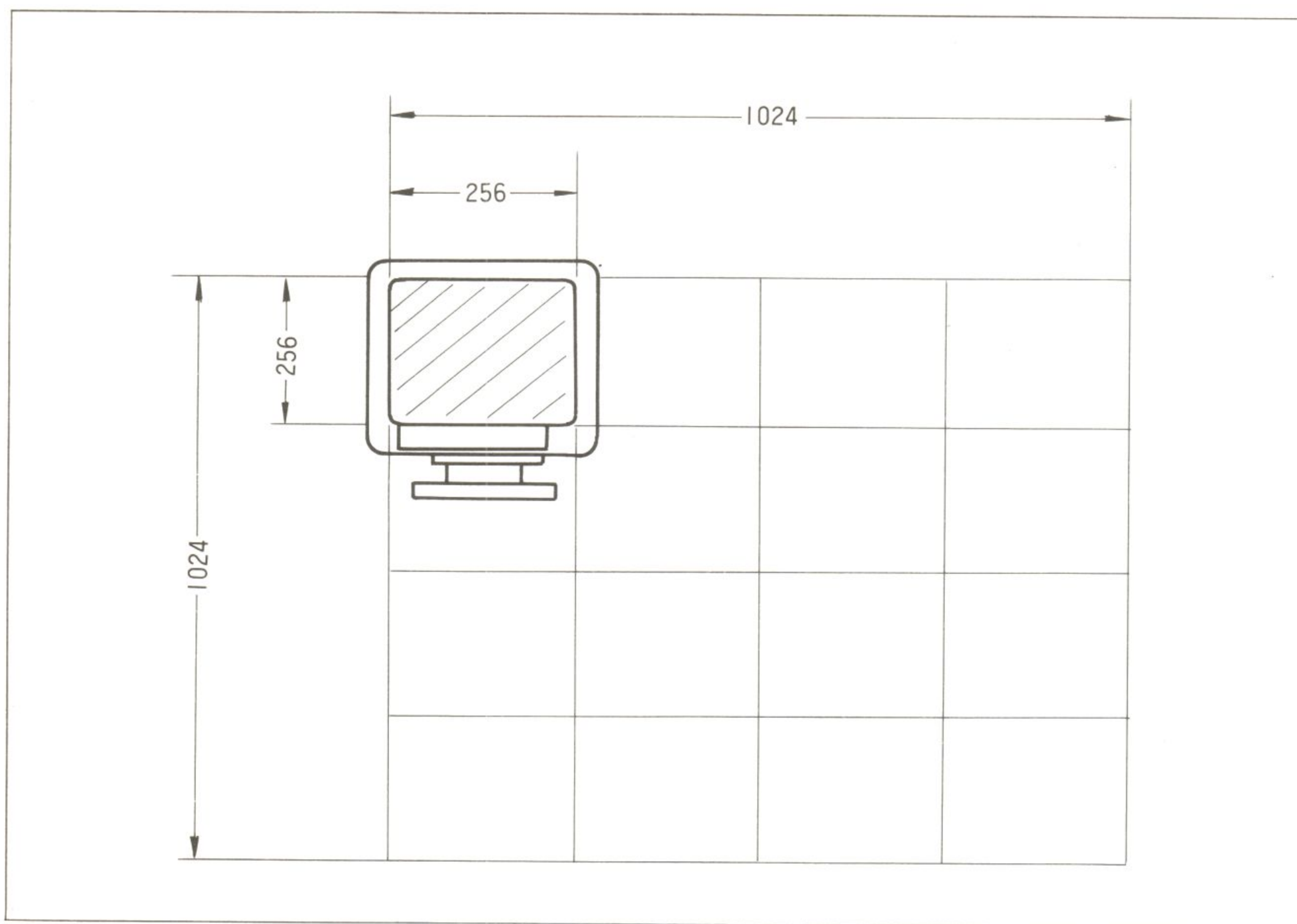
第３－33図　windowサンプルプログラム

10：コメント文
20：画面モードを768×512の16色モードに設定
30～40：パレットコード１，２に黄，水色をセット
50：表示範囲の左上座標を（100，100）右下座標を（300，300）に指定
60：表示範囲を画面に出力
70：表示範囲を水色の四角で描く。
80：定義関数en（　）を呼び出し実行
90～160：表示範囲を変えて50～80と同じ処理を行う。
180：定義関数en（　）の定義宣言
190：変数宣言
200～220：座標(384，256)を中心座標とし，半径を５ずつ増加させながら円を描く(半径は5～250まで)。
230：定義関数en（　）の終了宣言。この関数の呼び出された次の行へ戻る。

●表示画面と実画面

X68000は最大1024×1024という広さのグラフィック画面を持っていますが，ディスプレイが768×512までの広さしか表示できません。そこで，**表示画面サイズ**と**実画面サイズ**という，二つの画面サイズが存在します。

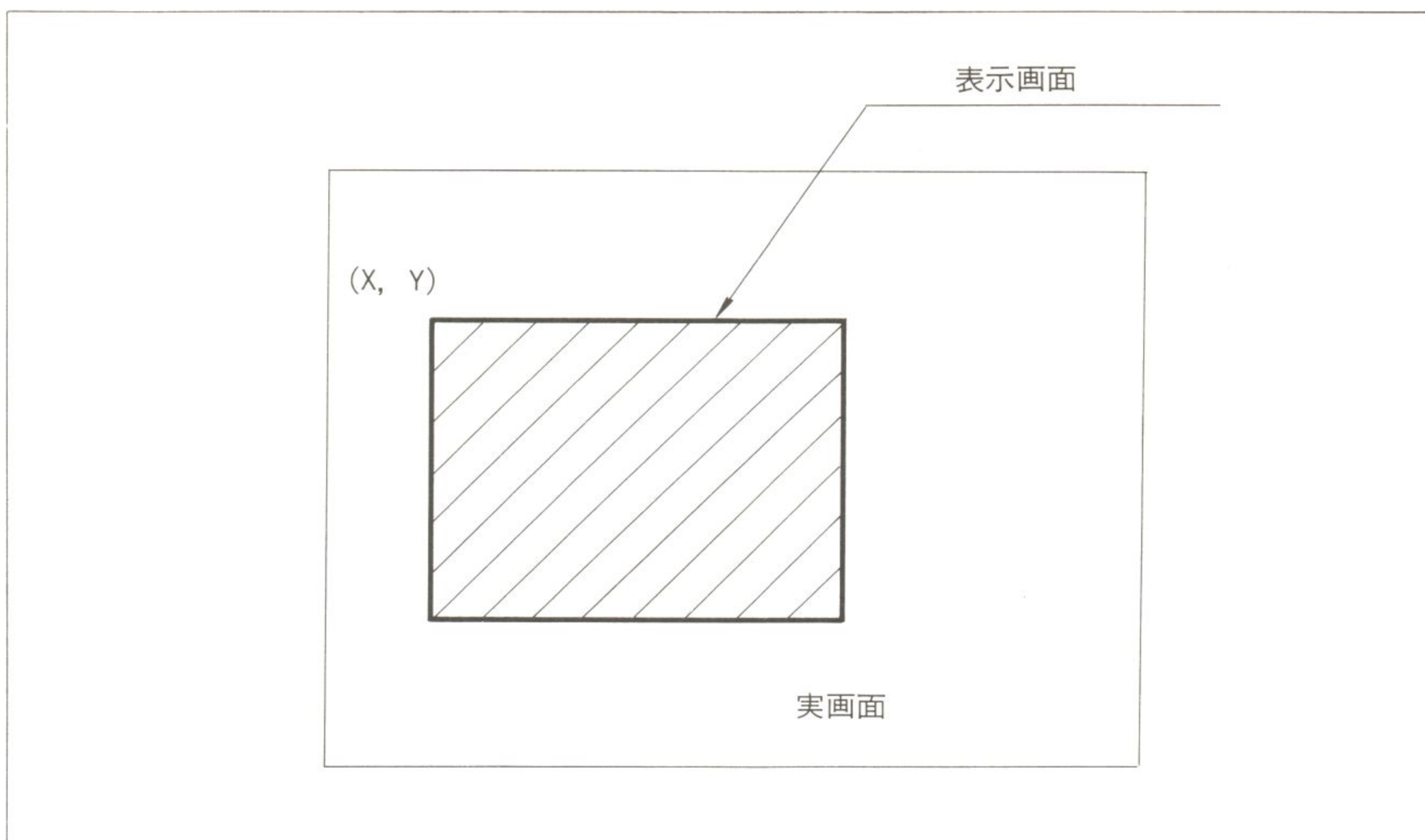
例えば，screen　0　，0・・・とした場合

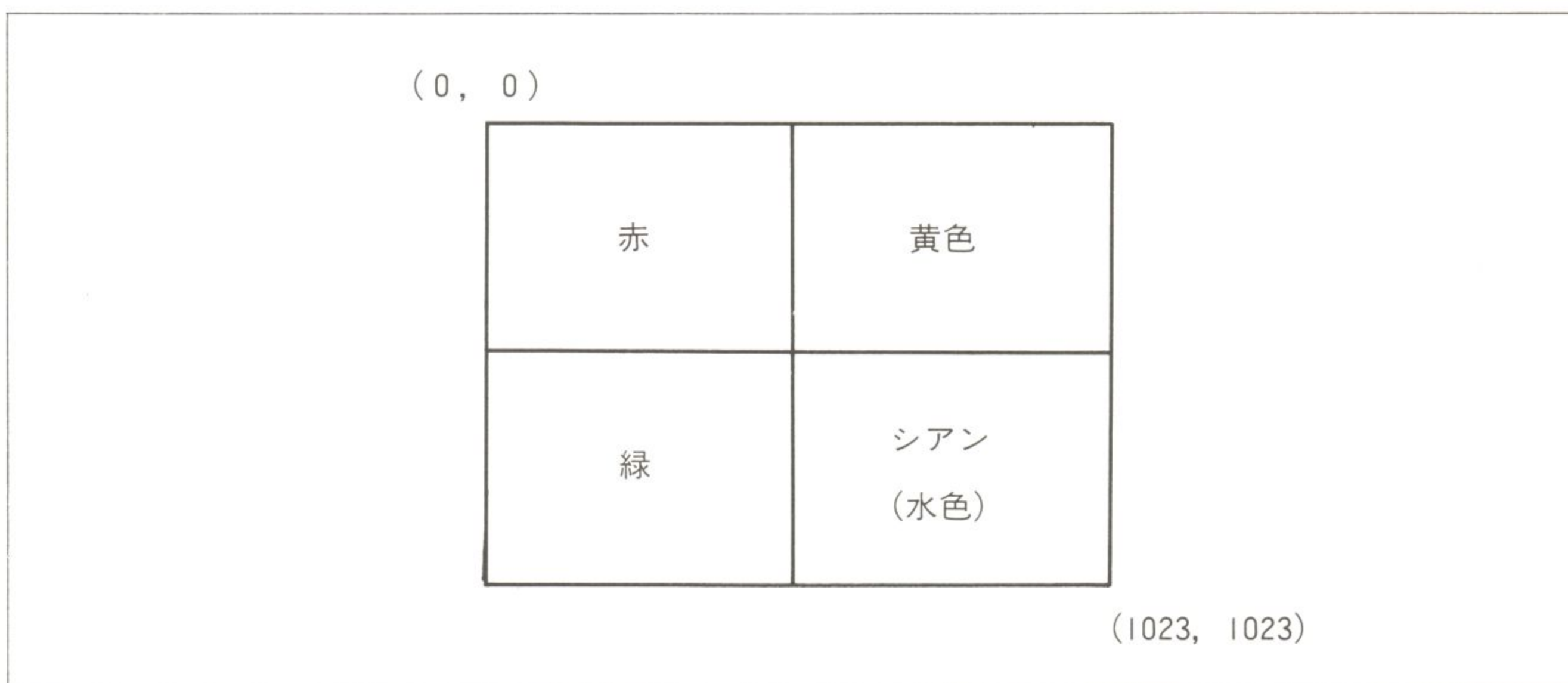
表示画面サイズは256×256で，実画面サイズは1024×1024となります。この時の表示画面と実画面は，**第３－34図**のような関係になっています。

第3－34図　実際に画面に表示できるのは斜線部分の範囲で，このサイズを表示画面サイズといいます

また，表示画面は実画面の中を1ドットごとに，自由自在にスクロールすることが出来ます。スクロールは球面スクロールといって，上下左右がつながったものでhome関数によって行われます。

【HOME】

書　式	home (pag, x, y)
引　数	char pag pag……グラフィックページ0～3（画面モードにより決まる） int x, y x……表示画面の実画面上の左上x座標 y……表示画面の実画面上の左上y座標

・x, yは画面モードにより0～511，または0～1023までの値
・screenを実行するとx, yは，0，0に設定される。
・**第3－35図参照**

第３－35図　home 関数における実画面と表示画面の関係

第３－37図は，実画面1024×1024を4等分（512×512）にし，各範囲を**第３－36図**のように色分けし，マウスを使って表示画面をスクロールさせるプログラムです。「百聞は一見にしかず」X68000の球面スクロールをぜひ味わってみて下さい。

第３－36図　実画面の色分け

```
10 /* home test
20 int x,y
30 screen 1,0,1,1
```

```
 40 window(0,0,1023,1023)
 50 mouse(0)
 60 mouse(4)
 70 palet(1,hsv(0,31,20))
 80 palet(2,hsv(32,31,20))
 90 palet(3,hsv(64,31,20))
100 palet(4,hsv(96,31,20))
110 fill(0,0,511,511,1)
120 fill(512,0,1023,511,2)
130 fill(0,512,511,1023,3)
140 fill(512,512,1023,1023,4)
150 while 1
160         mspos(x,y)
170         home(0,x,y)
180         locate 0,0:print"x=";x;"  y=";y
190 endwhile
200 end
```

第3－37図　home サンプルプログラム

10：コメント文

20：変数宣言

30：画面モード，表示画面サイズ512×512，実画面サイズ1024×1024，色65535色中16色モードに設定

40：クリッピングエリアを実画面サイズに設定

50～60：マウスの初期化及びマウスボタンの設定

70～100：パレットコード1～4に，赤，黄，緑，シアン（水色）を設定

110～140：実画面を4等分したサイズで，第3－36図のように色分けする（4ヶ所を各色の四角形で塗りつぶす)。

150：endwhile までの間を無限ループする。

160：このプログラムではマウスカーソルを表示していないが，マウスカーソルの指す座標をX，Yに読み込む。

170：160行で読み込んだX，Yの座標を，表示画面の実画面上の左上の座標とする。

180：左上座標X，Yの値を画面上に出力する。

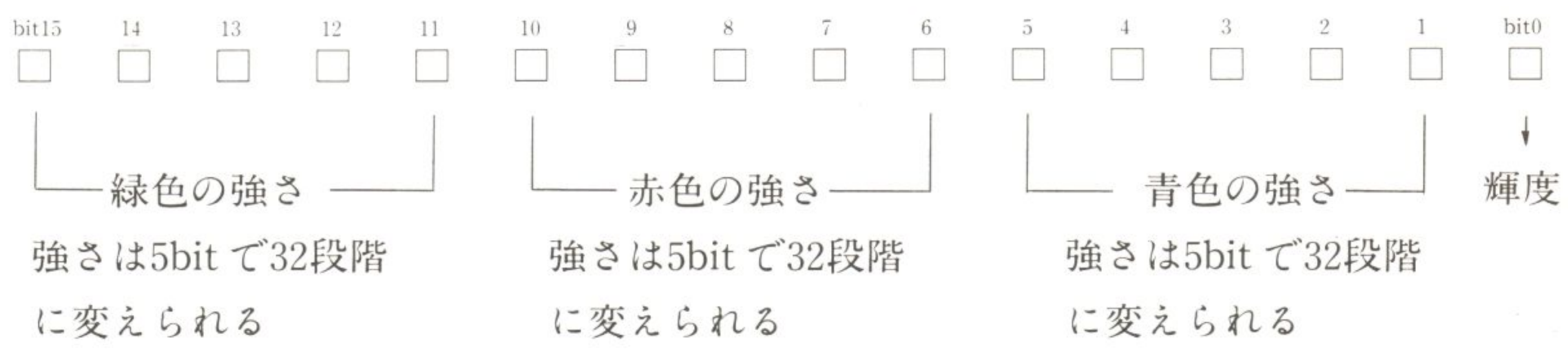
190：150行に戻る。

＊プログラムを終了するときは［break］キーを押して下さい。

(2) カラーコード

X68000では最大65536色使用することができます。X-BASICではこの色を0～65535の数字で表し，これをカラーコードと呼びます。カラーコードは，ただ単に数字と色を1対1に対応させただけではなく，色の並びとカラーコードにはある法則があります。この色とカラーコードの関係を図解すると，次のようになります。

16bitのカラーコードは，0～65535までの数値になります。

bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	bit0
□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
緑色の強さ					赤色の強さ					青色の強さ					輝度
強さは5bitで32段階に変えられる					強さは5bitで32段階に変えられる					強さは5bitで32段階に変えられる					

しかし，色を使うごとにこのようなビットの計算をしていたら大変です。X－BASICでは，このカラーコードを指定するための2種類の関係が用意されています。一つは「RGB」という関数で，光の三原色の原理で色を決めるものです。

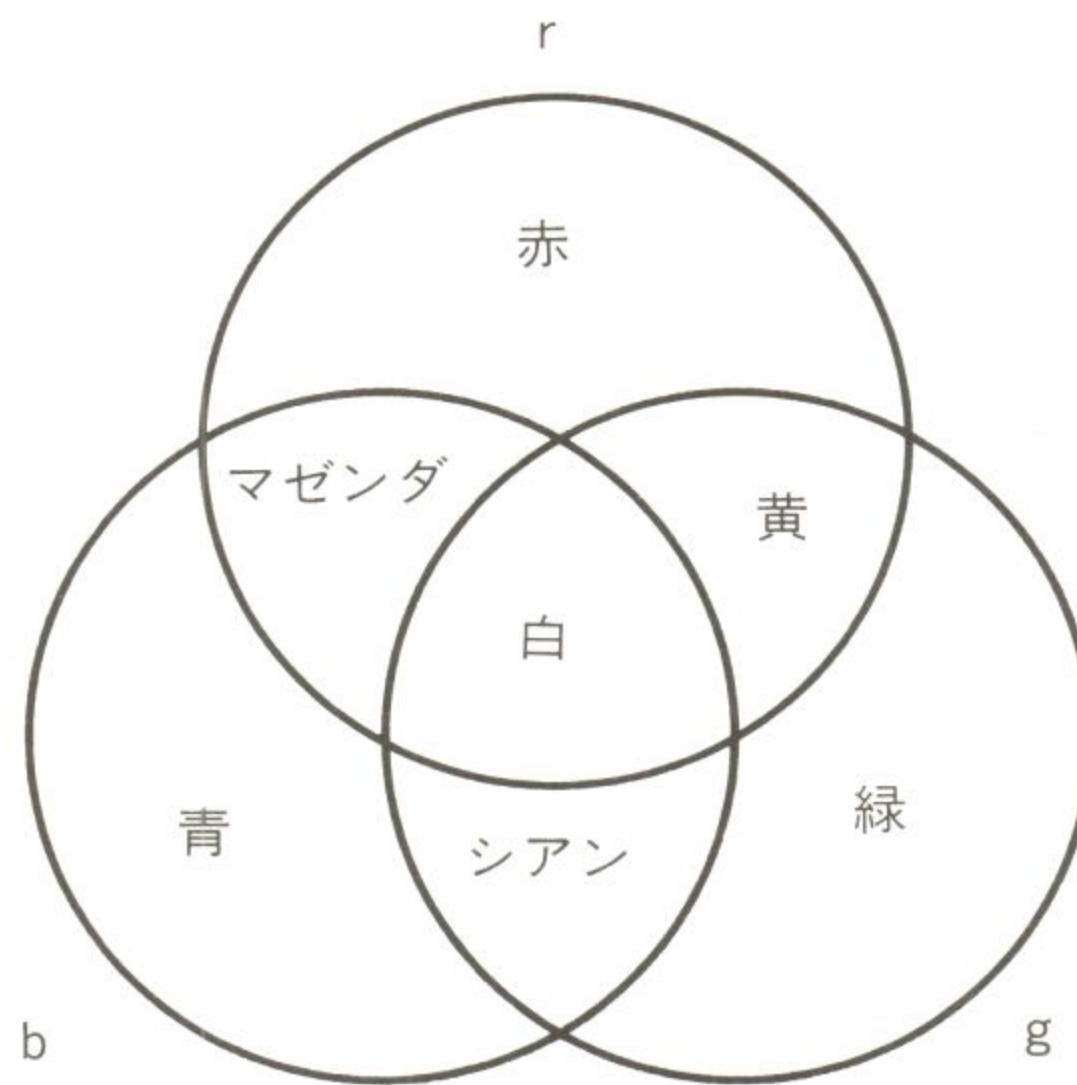
【RGB】

書　式	cl＝rgb（r，g，b）
引　数	int r，g，b r……赤成分の強さ g……緑成分の強さ b……青成分の強さ 数値が大きいほど明るくなります。
戻り値	int cl cl……カラーコード（0～65534の偶数値）

光の三原色というのは，赤，緑，青で，この混ざり合いによって**第3－38図**のような色になります。例えば，黄色のカラーコードを求めたい場合は，青成分を0にして赤と緑の成分の強さを同じ値にします。この値を大きくすれば明るく，小さくすれば暗い黄色になります。また，この関数では輝度成分は0となっていますので，最も明るい純粋な黄色のカラーコードは次のように求めることができます。

cl＝rgb（31，31，0）＋1

また，緑がかった黄色でしたら，その度合いに応じて緑成分を多くするか，赤成分を少なくすればよいのです。このように「RGB」関数は，絵の具を混ぜ合わせるような感覚で，色を求めることができます。**第3－39図**は，赤，緑，青成分の強さと輝度情報（0または1）をキーボード

r
赤
マゼンダ
黄
白
青
シアン
緑
b
g

第３－38図　光の三原色

から入力すると，「RGB」関数を使ってカラーコードを求め，その色で四角形を描きます。このプログラムを実行して，どの成分とどの成分を混ぜるとどのような色になるかを実験してみて下さい。

```
 10 /* rgb test
 20 screen 1,3,1,1
 30 int red,green,blue,kido,cl
 40 input"赤成分の強さ（０－３１）";red
 50 input"緑成分の強さ（０－３１）";green
 60 input"青成分の強さ（０－３１）";blue
 70 input"輝度（０－１）";kido
 80 cl=rgb(red,green,blue)+kido
 90 print "Color code = ";cl
100 fill(250,0,500,250,cl)
110 end
```

第３－39図　rgb サンプルプログラム

10：コメント文
20：画面モードを512×512ドットの65536色モードに設定
30：変数の宣言
40～60：各成分の強さをキーボードから変数に代入する。

70：輝度情報をキーボードから変数に代入する。

80：RGB関数を使って，カラーコードを求める。

90：求めたカラーコードを画面に出力する。

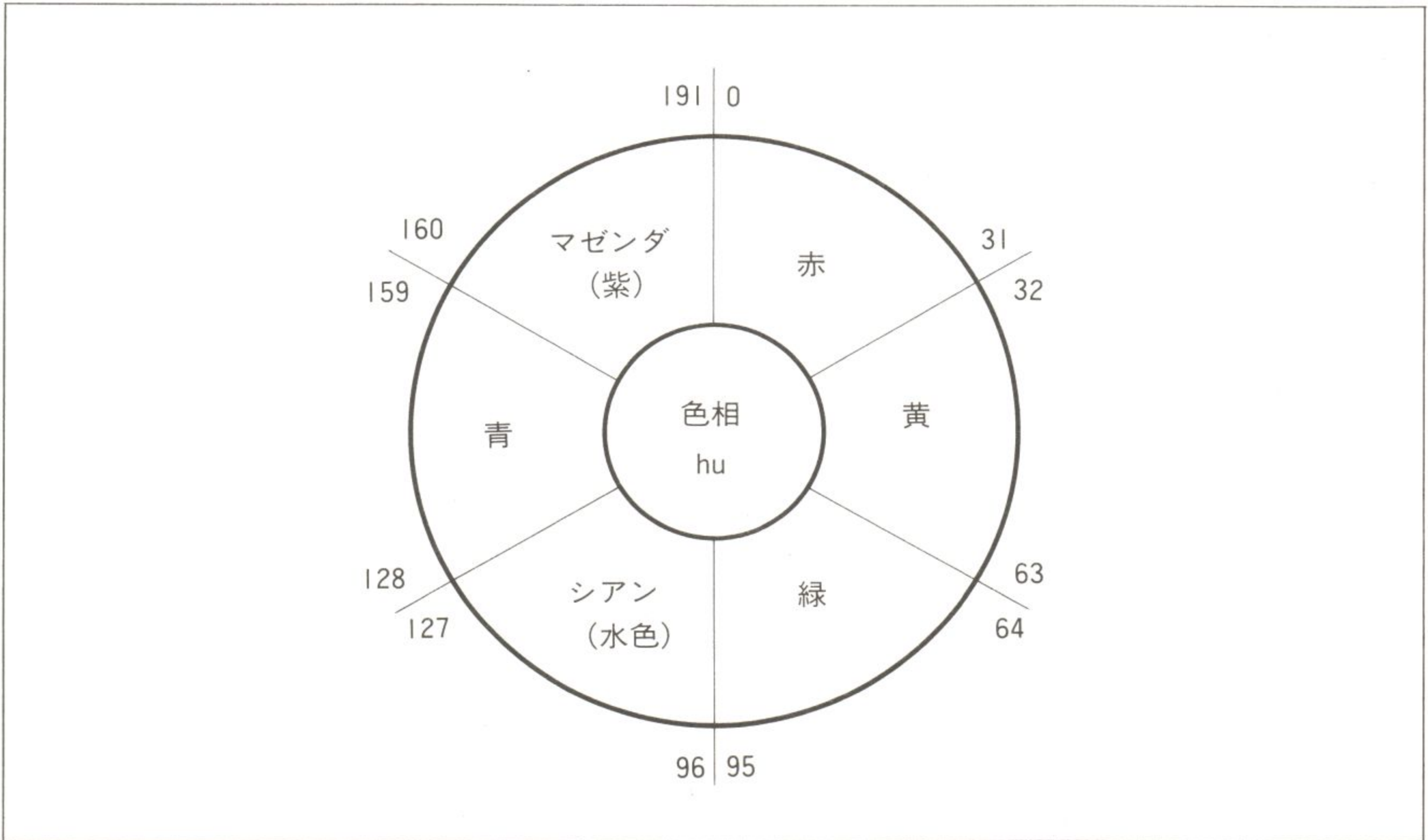
100：カラーコードによって表される色で，四角形を塗りつぶす。

二つめの関数は「HSV」です。これは，美術の授業等で聞いたことがあると思いますが「色相」，「飽和度」，「明るさ」という要素からカラーコードを求める関数です。

【HSV】

書　式	cl＝hsv（hu，sa，va）
引　数	int hu，sa，va hu……色相　（0～191） sa……飽和度（0～31） va……明るさ（0～31）
戻り値	int cl cl……カラーコード（0～65535の偶数値）

色相というのは，0～191までの数字を**第3－40図**のような色の配列に割り当てたものです。

例えば，色相で32というと完全な黄色ですが，63となると黄色の中で，最も緑に近い黄色になります。同様に，191はマゼンダの中で最も赤に近いものとなります。

第3－40図　色相

飽和度というのは，原色の純度を表します。飽和度が０ならば，色相で選んだ色は何も含まれていないことになり，31ならば色相で選んだ色（原色）となります。従って，飽和度の数値が多ければ多いほど原色に近づき，少なければ少ないほど白っぽい色になります。このことから，白系統の色（色，グレー，黒）等の場合は色相は関係なく，飽和度を０として次に説明する明るさのみをかえて作ることができます。

明るさは，同じ色の中でも暗い色と明るい色がありますので，その度合いを決めるものです。数値を多くすれば明るく，少なくすれば暗くなります。

また，この関数によって求められるカラーコードには，輝度情報は含まれていないので，求めたカラーコードに"１"をプラスするか，しないかでコントロールして下さい。

このように「HSV」関数は，カラーコードを求める場合きわめて感覚的に色を選択することができます。**第３－41図**は，色相，飽和度，明るさの三つをキーボードから入力すると，hsv関数を使ってカラーコードを求め，その色で四角形を描くプログラムです。

```
 10 /* hsv test
 20 int hu,sa,va,cl
 30 screen 1,3,1,1
 40 input"色相（０－１９１）";hu
 50 input"飽和度（０－３１）";sa
 60 input"明るさ（０－３１）";va
 70 cl=hsv(hu,sa,va)
 80 print "Color code = ";cl
 90 fill(250,0,500,250,cl)
100 end
```

第３－41図　hsvサンプルプログラム

10：コメント文

20：変数宣言

30：画面モードを512×512ドットの65536色モードにする。

40～60：各要素の数値をキーボードから変数に代入する。

70：hsv関数を使ってカラーコードを求める。

80：求めたカラーコードを画面に出力する。

90：求めたカラーコードによって表される色で，四角形を塗りつぶす。

（３）パレットコード

X-BASICでの色の使用のついては，前述のように65536色使えるのですが，画面モードによって65536色中の16色とか，65536色中の256色という制限があります。この考え方はユーザーの好き

な16色，または256色だけ取り出して使うというもので，16色モードの時は16個の絵の具の入れ物が，256色モードの時は256個の絵の具の入れ物が用意され，それぞれ０～15，０～255の番号が付けられています。これをパレットコードといいます。X68000では，色をすべてこのパレットコードで表現します。従って，65536色モードの時には，パレットコードは０～65535になります。このパレットコードにカラーコードを割り当てる命令が，「PALET」関数です。

【PALET】

書　式	palet (p，c)
引　数	int p，c p……パレットコード　０～255 c……カラーコード　０～65535

例えば，パレットコード０を赤色にしたい場合

palet（０，hsv（０，31，31））

とすれば良いわけです。**第３－42図**は，パレットコード１に赤，２に黄，３に緑，４にシアン，５に青，６にマゼンダ(hsv の色相の順) に設定し，四角形を描き何かキーを押した瞬間にパレットコードの設定を最初と逆にするプログラムです。

```
 10 /* palet test
 20 int aka,ki,midori,shian,ao,mazenda,i
 30 str space
 40 screen 1,2,1,1
 50 aka=hsv(0,31,31)
 60 ki=hsv(32,31,31)
 70 midori=hsv(64.31,31)
 80 shian=hsv(96,31,31)
 90 ao=hsv(128,31,31)
100 mazenda=hsv(160,31,31)
110 palet(1,aka)
120 palet(2,ki)
130 palet(3,midori)
140 palet(4,shian)
150 palet(5,ao)
```

```
160 palet(6,mazenda)
170 for i=0 to 5
180     fill(i*80,0,i*80+80,100,i+1)
190 next
200 space=inkey$()
210 palet(1,mazenda)
220 palet(2,ao)
230 palet(3,shian)
240 palet(4,midori)
250 palet(5,ki)
260 palet(6,aka)
270 end
```

第３－42図　palet サンプルプログラム

10：コメント文
20～30：変数宣言
40：画面モードを512×512ドット，256色モードに設定する。
50～100：hsv 関数を使い，各色のカラーコードをセットする。
110～160：パレットコード１～６に各色のカラーコードをセットする。
170～190：画面左からパレットコード１～６に対する色で，四角形を六つ描く。
200：何かキーが押されるまで待ち，キーが押されたら以下のプログラムを実行する。
210～260：パレットコード１～６に，110～160行で設定した逆の順でカラーコードをセットする。

このプログラムで注目していただきたいのが，四角形を画面に描くところは１回しかなく，何かキーを押した時には200行以下で現在描かれている四角形の色を palet 関数で変えているだけということです。

ここでもう一つ，パレットコードに関係する関数を説明しておきます。それは，「POINT」関数です。これはグラフィック画面上のパレットコードを求める関数です。

【POINT】

書　式	pa＝point（x，y）
引　数	int x，y x……グラフィック画面のｘ座標　window 関数で， y……グラフィック画面のｙ座標　指定された範囲のみ

戻り値	int pa pa……パレットコード（0～65535）

第3－43図は，マウスを左クリックした時の座標のパレットコードを求めるプログラムです。入力する時は，第3－42図の200行からを変更して，入力して下さい。終了する時は右クリックします。

```
200 /* point test
210 int mx,my,x,y,bl,br,pa
220 mouse(0)
230 mouse(1)
240 mouse(4)
250 while br=0
260      msstat(mx,my,bl,br)
270      if bl=-1 then pal()
280 endwhile
290 mouse(0)
300 end
310 func pal()
320     mspos(x,y)
330     pa=point(x,y)
340     locate 0,8
350     print"座標";x;",";y;"のパレットコードは、";pa
360 endfunc
```

第3－43図　paint サンプルプログラム

10：コメント文
20～30：変数宣言
40：画面モードを512×512ドット，256色モードに設定する。
50～100：hsv 関数を使い各色のカラーコードを求め変数に代入する。
110～160：パレットコード1～6に各色のカラーコードをセットする。
170～190：画面左からパレットコード1～6に対する色で，四角形を六つ描く。
200：コメント文
210：変数宣言
220：マウスの初期化

230：マウスカーソルの表示
240：ソフトキーボードの表示
250：マウスの右ボタンが押されるまで，280行の endwhile 間をループ
260：マウスのボタンが押されたかを見る。
270：もし左ボタンが押されていたら，定義関数 pal（　）に行く。
280：while の行（250行）に戻る。
290：右ボタンが押されると，while～endwhile のループを抜け，この行にきてマウスの初期化を行う。
300：プログラムの終了
310：定義関数 pal（　）の定義宣言
320：マウスカーソルのある座標を，変数 X，Y に読み込む。
330：座標 X，Y のドットのパレットコードを求め，pa に代入する。
340～350：p a の値を画面に出力する。
360：定義関数終了宣言。この関数が呼び出された行の次の行（280行）に戻る。

（4）描画

この項からは，実際にグラフィック画面に描画を行う関数群を説明していきます。

〈点の描画〉

グラフィック画面に点（ドット）を表示する関数が「PSET」関数です。

【PSET】

書　式	pset（x，y，pa）
引　数	int x，y，pa x……表示するドットのx座標　　－32768～32767まで y……表示するドットのy座標　　－32768～32767まで クリッピングエリアは，window 関数で指定された範囲 pa……パレットコード　画面モードにより 0～65535

第3－44図は，マウスを左クリックした時のマウスカーソルが指す座標のドットを緑色で表示するプログラムです。ゆっくり左ドラッグ（左クリックしたままでマウスを動かす）して見て下さい。

```
10 /* pset test
20 int x,y,mx,my,br,bl,cl
30 screen 1,2,1,1
```

```
 40 cl=hsv(64,31,31)
 50 palet(1,cl)
 60 mouse(0)
 70 mouse(1)
 80 mouse(4)
 90 while br=0
100         msstat(mx,my,bl,br)
110         if bl=-1 then psetting()
120 endwhile
130 mouse(0)
140 end
150 func psetting()
160       mspos(x,y)
170       pset(x,y,1)
180 endfunc
```

第3－44図　pset サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを512×512ドットの256色モードに設定
40：hsv 関数で緑色のカラーコードを求め，変数に代入
50：パレットコード１に先程のカラーコードを求め，変数に代入
60～140：第３－43図の220行～300行までと同じ。ただし，左クリック ON の時に呼び出す定義関数は"psetting（　）"です。
150：定義関数 psetting（　）の定義宣言
160：マウスカーソルのある座標をｘ，ｙに読み込む。
170：座標ｘ，ｙにパレットコード１のドットを表示する。
180：定義関数終了宣言。この関数が呼び出された次の行（120行）に戻る。

〈線の描画〉

グラフィック画面に直線を引く関数が「LINE」関数です。

【LINE】

書　式	line（x1，y1，x2，y2，ls）
引　数	int　x1，y1，x2，y2，pa，ls

	ｘ１……始点のｘ座標 ｙ１……始点のｙ座標 ｘ２……終点のｘ座標 ｙ２……終点のｙ座標 ｐａ……パレットコード（線の色） ｌｓ……ラインスタイル

・x1，y1，x2，y2は，－32768～32767までの値
・クリッピングエリアはwindow関数で指定された範囲
・Paは画面モードにより０～65535
・ラインスタイルは０～65535（&H0000～&HFFFF）

ラインスタイルというのは，点線や破線などを16ビットのビットパターンで表したもので，直線ならば，

　　１１１１　１１１１　１１１１　１１１１
　　　F　　　　F　　　　F　　　　F

このパターンの繰り返しなので，ラインスタイルは&HFFFF，点線ならば，

　　１１００　１１００　１１００　１１００
　　　C　　　　C　　　　C　　　　C

なので，&HCCCCとすればよいわけです。以下に，主なラインスタイルのパターンを示しますので参考にして下さい。

点線	&HFF00，&HCCCC，&HAAA	－－－－－－－－
一点鎖線	&HFFCC，&HE4E4	－・－・－・－
二点鎖線	&HFF24	－・・－・・－

ラインスタイルを省略した場合は，実線（&HFFFF）となります。

第３－45図は，原点（画面上の左上）からマウスを左クリックしたところの座標まで，指定のラインスタイルで黄色の線を引くプログラムです。ドラッグするとおもしろいと思います。実行するとラインスタイルを聞いてくるので，&H○○○○の形で入力して下さい。右クリックすると終了します。

```
10 /* line test
20 int cl,ls,mx,my,bl,br,x,y
30 screen 1,2,1,1
40 cl=hsv(32,31,31)
50 palet(1,cl)
60 input"ラインスタイルを入力して下さい。",ls
70 mouse(0)
```

```
 80 mouse(1)
 90 mouse(4)
100 while br=0
110       msstat(mx,my,bl,br)
120       if bl=-1 then lin()
130 endwhile
140 mouse(0)
150 end
160 func lin()
170      mspos(x,y)
180      line(0,0,x,y,1,ls)
190 endfunc
```

第３－45図　line サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを，512×512ドットの256色モードに設定
40：hsv 関数で黄色のカラーコードを求め，変数に代入する。
50：パレットコードに先程のカラーコードをセットする。
60：ラインスタイルをキーボードから入力し，変数に代入する。
70～150：第３－43図の220行～300行と同じ。ただし，左クリックした時に呼び出される関数は lin（　）
160：定義関数 lin（　）の定義宣言
170：マウスカーソルのある座標を，x2．y2に読み込む。
180：ラインの始点座標（0，　0）から，終点座標（x1．y1）（マウスカーソルの座標）まで，先程入力したラインスタイルで黄色の線を引く。
190：定義関数終了宣言。この関数が，呼び出された次の行（130行）に戻る。

<四角形の描画>

四角形に関する関数は二つあり，一つは，「BOX」，もう一つが「FILL」です。前者は線によって四角形を描くもので，後者は中を塗りつぶした四角形を描きます。

【BOX】

書　式	BOX（x1，y1，x2，y2，pa，ls）
引　数	int　x1，y1，x2，y2，pa，ls

ｘ１……始点ｘ座標
ｙ２……始点ｙ座標
ｘ１……終点ｘ座標
ｙ２……終点ｙ座標
ｐａ……パレットコード（線の色）
ｌｓ……ラインスタイル

・x1，y1，x2，y2は－32768～32767までの値
・クリッピングエリアはwindow関数で，指定された範囲
・paは画面モードにより０～65535
・ラインスタイルは０～65535（&H0000～&HFFFF）

第３－46図は，原点（画面上の左上）からマウスを左クリックしたところの座標を終点としたシアン（水色）の四角形を，指定されたラインスタイルで描くものです。

```
 10 /* box test
 20 int cl,ls,mx,my,bl,br,x,y
 30 screen 1,2,1,1
 40 cl=hsv(96,31,31)
 50 palet(1,cl)
 60 input"ラインスタイルを入力して下さい。",ls
 70 mouse(0)
 80 mouse(1)
 90 mouse(4)
100 while br=0
110      msstat(mx,my,bl,br)
120      if bl=-1 then lin()
130 endwhile
140 mouse(0)
150 end
160 func lin()
170      mspos(x,y)
180      box(0,0,x,y,1,ls)
190 endfunc
```

第３－46図　boxサンプルプログラム

このプログラムは，第３－44図とほとんど同じです。どこが違うかは，自分で探してみて下さい。次は，指定の色で塗りつぶした四角形を描く「FILL」関数です。

【FILL】

書　式	fill (x1, y1, x2, y2, pa)
引　数	int　x1, y1, x2, y2, pa ｘ１……始点ｘ座標 ｙ２……始点ｙ座標 ｘ１……終点ｘ座標 ｙ２……終点ｙ座標 ｐａ……パレットコード（塗りつぶす色）

・x1, y1, x2, y2は－32768～32767までの値
・クリッピングエリアはwindow関数で，指定された範囲
・paは画面モードにより０～65535

第３－47図は，マウスを左クリックした時のマウスカーソルの座標を始点とし，縦20ドット，横20ドットの塗りつぶした四角形を描くプログラムです。なお，塗りつぶしの色は，hsv関数の飽和度をだんだん増加させることによって，変化させています。

```
10 /* fill test
20 int cl,mx,my,bl,br,x,y
30 int i,j
40 screen 1,2,1,1
50 mouse(0)
60 mouse(1)
70 mouse(4)
80 while br=0
90       msstat(mx,my,bl,br)
100      if bl=-1 then fil()
110 endwhile
120 mouse(0)
130 end
140 func fil()
160      i=i+1
```

```
170      if i>31 then i=1
175      cl=hsv(0,i,31)
180      palet(i,cl)
190      mspos(x,y)
200      fill(x,y,x+20,y+20,i)
210      for j=0 to 100
220      next
230 endfunc
```

第３－47図　fill サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを512×512ドットの256色モードに設定
40～120：第３－43図の220行～300行と同じ。ただし，左クリックした時に呼び出される関数は，fillset（　）
130：定義関数の fillset（　）の定義宣言
140：変数paを一つ増加させる（このプログラムでは，変数paをパレットコードの指定，及び飽和度の値として使用している）。
150：もし pa が31を越えるとエラーとなるので,(飽和度の設定範囲は，０～31）31を越えたら pa＝１とする。
160：hsv 関数でカラーコードを求める。
170：求めたカラーコードをパレットコード pa にセットする。
180：マウスカーソルの座標を x1，y1に読み込む。これが始点座標となる。
190～200：x1，y1の値に20ずつプラスして，x2，y2に代入する。これが終了座標となる。
210：パレットコード pa で塗りつぶした四角形を描く。
220～230：次の処理のための時間つぶし（空ループ)。
240：定義関数終了宣言。この関数が呼び出された次の行（100行）に戻る。

<円の描画>

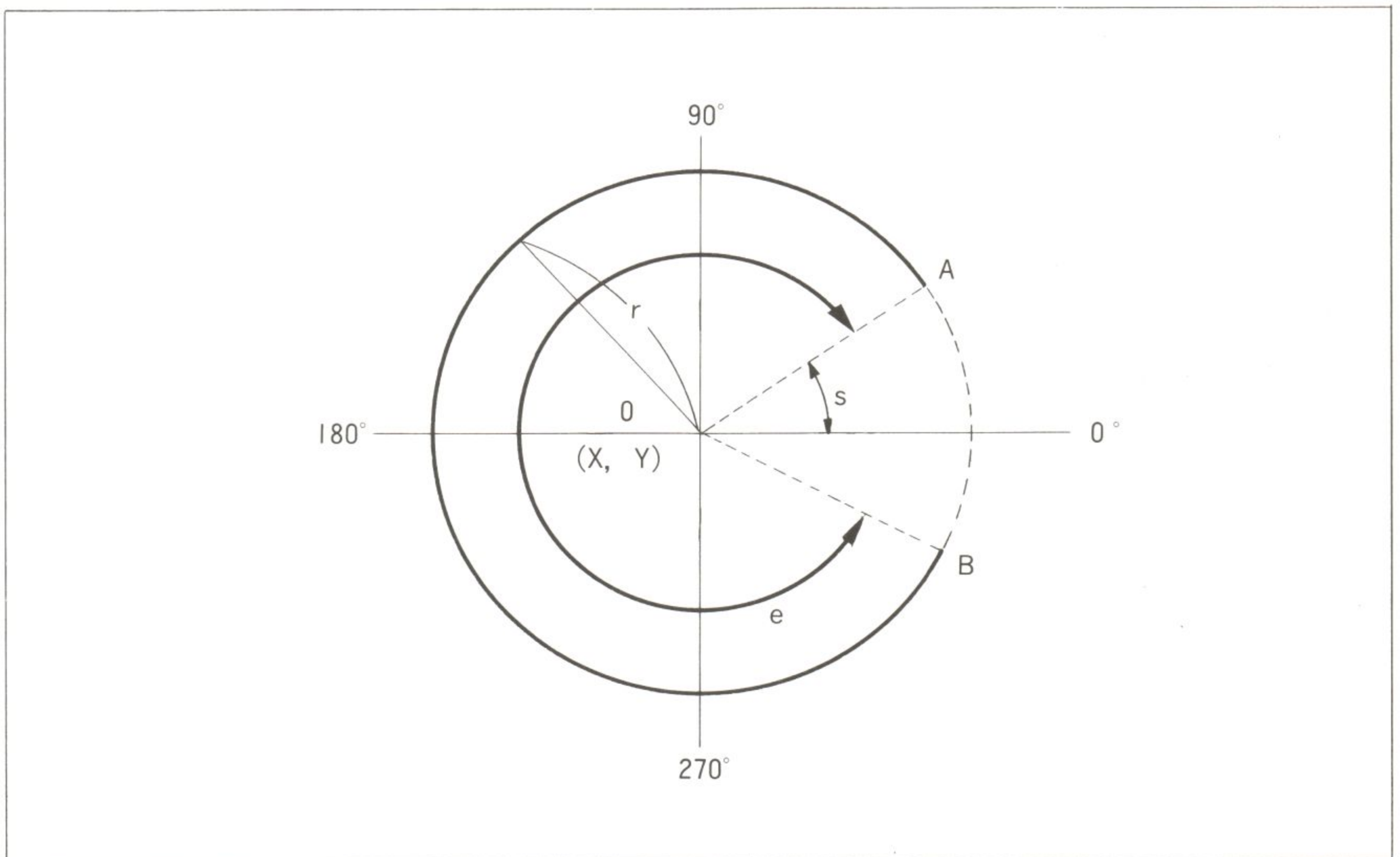
グラフィック画面に円を描く関数が「CIRCLE」関数です。

【CIRCLE】

書　式	circle（x，y，r，pa，s，e，hv)
引　数	int　x，y，r，pa，s，e，hv x……円の中心の x 座標

	ｙ……円の中心のｙ座標 ｒ……円の半径 ｐａ…パレットコード（円の色） ｓ……開始角度 ｅ……終了角度 ｈｖ…偏平率

・ｘ，ｙは－32768～32767までの範囲
・ｒは０～32767までの値
・クリッピングエリアはwindow関数で指定された範囲
・paは画面モードにより０～65535まで
・ｓ，ｅは－360°～360°の値
・hvは０～32767

開始角度ｓと終了角度ｅというのは，**第３－48図**のような関係になっています。

第３－48図　開始角度と終了角度

従って，円を描くためにはｓ＝０，ｅ＝360とすればよいわけです。また，ｓとｅの値に－１°～－360°の値を使用すると，図の円の中心０からＡとＢを線で結んだ扇形を描くことができます。

偏平率hvは256より小さければ横長の楕円形となり，256より大きければ縦長の楕円形となります。正円とする場合は，hv＝256とします，しかし，ディスプレイの関係上，真円にはなりません。

第３－49図は，circle関数の各パラメータをキーボードから入力すると，パラメータどおりの円

(弧，扇形) を描き，最後に入力したパラメータをcircle関数の書式で出力するものです。色は100行で黄色にセットしてあります。いろいろな円を描いてみて，使い方を覚えて下さい。

```
 10 /* circle test
 20 int x,y,r,pa,s,e,hv,cl
 30 screen 1,2,1,1
 40 input"中心X座標";x
 50 input"中心Y座標";y
 60 input"半径r    ";r
 70 input"開始角度s";s
 80 input"終了角度e";e
 90 input"偏平率hv";hv
100 cl=hsv(32,31,31)
110 palet(1,cl)
120 circle(x,y,r,1,s,e,hv)
130 print"circle(";x;",";y;",";r;",pa,";s;",";e;",";hv;")"
140 end
```

第3－49図　circleサンプルプログラム

```
 30 screen 2,0,1,1
```

第3－50図　変更部分

10：コメント文
20：変数宣言
30：画面モードを512×512ドットの256色モードに設定
40～90：各パラメータをキーボードから入力し，各変数に代入する。
100：hsv関数で黄色のカラーコードをセットする。
110：パレットコード1に先程のカラーコードをセットする。
120：各パラメータどおりの黄色の円を描く。
130：circle関数の書式で画面に出力する。この時，パレットコードの部分はpaとして出力する。

また，30行を**第3－50図**のように変更すれば，768×512ドットの時の円の描写ができます。これにより，512×512の時と，768×512の時の円の感じをつかんで下さい。

〈ペイント〉

四角形を塗りつぶす関数はfill関数でしたが，circle関数で描いた円や，line関数を使って描いた三角形などを塗りつぶす時に使用するのが「PAINT」関数です。

【PAINT】

書　式	paint（x，y，pa）
引　数	int　x，y，pa x……塗りつぶしたい領域内の任意の点のx座標 y……塗りつぶしたい領域内の任意の点のy座標 ｐａ…パレットコード（塗りつぶす色）

・x，yは－32768～32767までの値
・paは画面モードにより，０～65535まで

領域というのは，ある色で囲まれた場所で，その中の点（どこでもよい）の座標を与えてやれば，パレットコードで指定した色で塗りつぶしてくれます。

第3－51図は，実行させるといろいろなパターンの図形があらわれるので，塗りつぶしたい領域の中にマウスカーソルをもっていき，左クリックすることで青色に塗りつぶします。このプログラムで，領域という意味を確認してみて下さい。

```
 10 /* paint test
 20 int cl,ls,mx,my,bl,br,x,y
 30 screen 1,2,1,1
 40 palet(1,hsv(0,31,31))
 50 palet(2,hsv(128,31,31))
 60 palet(3,hsv(64,31,31))
 70 circle(50,100,30,1,0,360,256)
 80 circle(150,100,30,1,0,350,256)
 90 box(20,200,100,300,1)
100 box(320,200,400,300,2)
110 fill(180,200,280,300,1)
120 fill(320,200,370,300,1)
130 fill(371,200,420,300,3)
140 mouse(0)
```

```
150 mouse(1)
160 mouse(4)
170 while br=0
180       msstat(mx,my,bl,br)
190       if bl=-1 then painting()
200 endwhile
210 mouse(0)
220 end
230 func painting()
240      mspos(x,y)
250      paint(x,y,2)
260 endfunc
```

第３－51図　paint サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを，512×512ドットの256色モードに設定
40～60：パレットーコード１～３に，赤，青，緑をセットする。このように，palet 関数の中に，いきなり hsv 関数を書いて，カラーコードを指定することもできる。
70～130：各サンプル図形の描画
140～220：第３－43図の220行～300行と同じ。ただし，左クリックした時に呼び出される関数は，painting（　）
230：定義関数 painting（　）の定義宣言
240：マウスカーソルのある座標をｘ，ｙに読み込む。
250：マウスカーソルのある座標ｘ，ｙが含まれている領域内を，パレットコード２の青色で塗りつぶす。
260：定義関数終了宣言。この関数が呼び出された次の行（200行）に戻る。

〈文字の描画〉

文字を表示できるのは，テキスト画面だけでなく「SYMBOL」関数を使うことでグラフィック画面にも表示できます。

【SYMBOL】

書　式	symbol（x，y，st，h，v，mo，pa，an）
引　数	int x，y，pa

x……始点x座標
y……始点y座標
pa…パレットコート（文字の色）
・x，yは－32768～32767までの値
・paは画面モードにより，０～65535まで
・クリッピングエリアについては，window関数で指定された範囲

str　　st
　　st…表示する文字例（文字式でも可）

shar　　h，v，mo，an
　　h……横方向の倍率
　　v……縦方向の倍率
　　mo…文字フォントの種類
　　　mo＝０のとき　6　12
　　　mo＝１のとき　8　16
　　　mo＝２のとき　12　24
　　an…表示文字列の回転角度
　　　an＝０のとき　0°
　　　an＝１のとき　90°
　　　an＝２のとき　180°
　　　an＝３のとき　270°

第3－52図は，始点x，y座標，表示文字列，縦横の倍率，文字フォントの種類を入力すると，始点x，y座標を中心に各回転角度で，表示文字列を緑色で描くプログラムです。なお，この時始点x，y座標（回転の中心となる）に水色でドットを表示します。

```
 10 /* symbol test
 20 int x,y,i
 30 str st
 40 char h,v,mo,an
 50 screen 1,2,1,1
 60 input"始点X座標";x
 70 input"始点Y座標";y
 80 input"表示文字列";st
 90 input"横の倍率  ";h
100 input"縦の倍率  ";v
110 input"文字種    ";mo
```

```
120 an=0
130 palet(1,hsv(64,31,31))
140 palet(2,hsv(96,31,31))
150 pset(x,y,2)
160 for  i=0 to 3
170     symbol(x,y,st,h,v,mo,1,an)
180     an=an+1
190 next
200 end
```

第３－52図　symbol サンプルプログラム

10：コメント文
20～40：変数宣言
50：画面モードを512×512ドットの256色モードに設定
60～110：各パラメータをキーボードから入力し，各変数に代入する。
120：表示文字列の回転角度の初期値を０（０°）にする。
130：パレットコード１に緑色をセット
140：パレットコード２に水色をセット
150：始点座標ｘ，ｙに水色のドットを表示する。
160：190行の next の間を４回ループする（０～３)。
170：入力されたパラメータで文字を緑色で描く。回転角度は an の値で決める。
180：an の値を一つ増加させる。
190：next 文

<グラフィック画面の消去>

テキスト画面を「CLS」という命令で消去できるように，グラフィック画面を消去する関数「WIPE（　)」があります。この関数は，アクティブグラフィックページ（「apage」関数で指定されているページ）について消去を行います。

【WIPE】

書　式	wipe（　)
引　数	なし

第３－53図は，画面モードでグラフィック画面を４ページ確保して，４ページ各々を違う色で塗りつぶした後，何かキーを押すごとに０ページ目から順番に，wipe 関数を使って消去していくプログラムです。

```
 10 /* wipe test
 20 int pag,i,j
 30 str space,mes1,mes2
 40 mes1="これは、"
 50 mes2="ページです。"
 60 dim str pn(3)={"０","１","２","３"}
 70 screen 1,1,1,1
 80 vpage(15)
 90 palet(1,hsv(0,31,31))
100 palet(2,hsv(32,31,31))
110 palet(3,hsv(64,31,31))
120 palet(4,hsv(96,31,31))
130 palet(5,hsv(0,0,0))
140 for pag=0 to 3
150    apage(3-pag)
160    pa=pag+1
170    fill(0,0,512,512,pa)
180    symbol(100,250,mes1+pn(3-pag)+mes2,1,1,2,5,0)
190    for j=0 to 1000
200    next
210 next
220 pag=0
230 while pag<>4
240      space=inkey$
250      apage(pag)
260      wipe()
270      pag=pag+1
280 endwhile
290 end
```

第3−53図　wipe サンプルプログラム

10：コメント文

20〜30：変数宣言
40〜50：文字変数に，コメントのための文字データを代入する。
60：文字型の配列を四つ確保し，０，１，２，３という文字データを代入する。
(pa（0）=”０”，pa（1）=”１”，・・・pa（3）=”３”)
70：画面モードを512×512ドットの256色モードに設定
これでグラフィックページは，０〜３の４ページが使用可能となります。
80：vpage 関数で０〜３ページすべてを表示 ON に設定する。
90〜130：パレットコード１〜５に各カラーコードをセットする。
140：210行の next の間を４回ループする。この時，pag は０〜３まで変化する。
150：apage 関数でアクティブページの設定を行う。この場合，３，２，１，０ページの順で設定される。
160：パレットコード pa を１，２，３，４の順で設定する。
170：512×512の範囲を pa で表されるパレットコードで塗りつぶす。
180：「symbol」関数で各ページを表すメッセージを描く。
190〜200：時間調整のための空ループ
210：next 文
220：ページを表す変数 pag を０にする。
230：pag が３より大きくなるまで，280行の endwhile 間をループ
240：何かキーが押されるまで待つ。押されたら，次の行へ進む。
250：apage 関数でアクティブページの設定を行う。この場合，０，１，２，３ページの順で設定される。
260：アクティブグラフィックページの消去を行う。
270：ページを表す pag の値を１増加させる。
280：endwhile 文。230行に戻る。

(５) その他の関数

画面上に描いたグラフィックを違う場所にコピーしたり，ディスクに保存したりするための関数が get と put です。get と put を使用する場合には，あらかじめ配列を用意しておき，その配列を使ってデータをやりとりします。考え方としては，get でデータを配列に読み込み，put で配列のデータをグラフィック画面に書き込むようになります。従って，この配列のデータを fwrite 関数を使ってディスクに書き込めば，グラフィック画面の保存が可能になります。

【GET】

書式	get（x1，y1，x2，y2，na）
引数	int　x1，y1，x2，y2 x1…始点の x 座標，y1…始点の y 座標，x2…終点の x 座標，y2…終点の y 座標 na…数値型の一次元配列名（パターンデータを格納する）

・x1，y1，x2，y2は，window 関数で指定された範囲内の値を取ることができます。

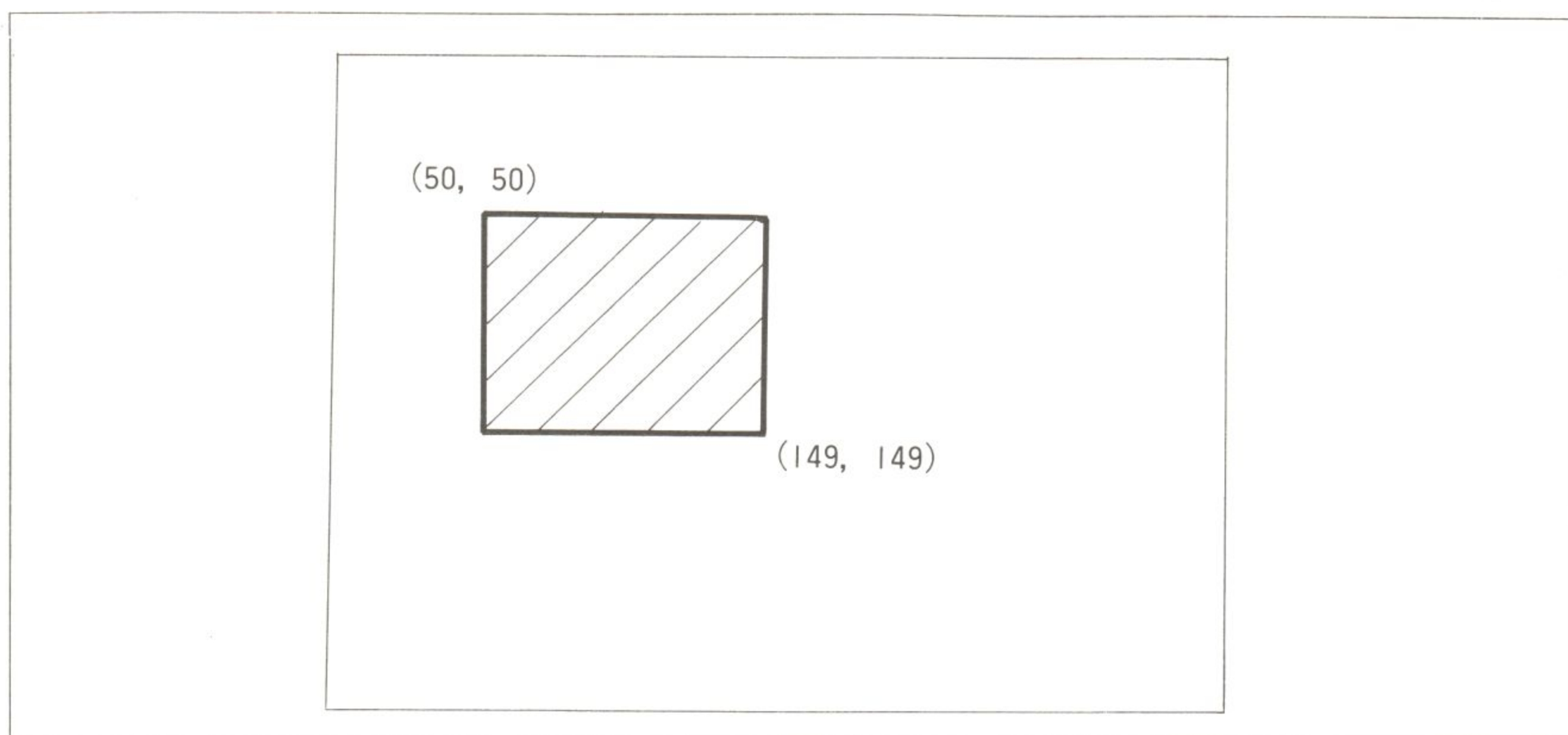
・配列 na へのドットパターンの読み込みかたは（x1，y1)，(x1+1，y1)，…（x2，y1)，(x1，y1＋1)，（x1＋1，y1＋1)，…（x2，y2）となります。

また，画面モードによって，配列に読み込むデータが変化します。

実画面1024×1024，16色モードの場合は，一つのドットを4ビットで表現しているので，読み込む配列の型によって次のようになります。

・char 型配列…………一つの配列の中に2ドットのデータが入る

・int 型配列……………一つの配列の中に8ドットのデータが入る

・float 型配列…………一つの配列の中に16ドットのデータが入る

第3－54図　画面上に四角のグラフィックを描く

例えば，**第3－54図**の領域のパターンを読み込む場合，ドット数は縦100ドット×横100ドット＝10000ドット分の配列を用意することが必要です。この時，

・char 型なら　10000／2＝　5000ヶ

・int 型なら　　10000／8＝　625ヶ

・float 型なら　10000／16＝　313ヶ

の配列を用意します。

実画面512×512，16色モードの場合は1024×1024，16色モードと同じですが，apage 関数で指定されたページに対してのみ有効です。

実画面512×512，256色モードの場合は1ドットを1バイトで表現しているので，

・char 型…………一つの配列の中に1ドットのデータが入る

・int 型……………一つの配列の中に4ドットのデータが入る

・float 型…………一つの配列の中に8ドットのデータが入る

この時も apage 関数で指定されたページに対してのみ有効です。

実画面512×512，65536色モードの場合は，1ドットを2バイトで表現しているので，

・char 型…………二つの配列の中に1ドットのデータが入る

・int 型……………一つの配列の中に2ドットのデータが入る

・float 型…………………一つの配列の中に４ドットのデータが入る

10000ドットの領域を char 型配列に読み込むためには，この画面モードの場合には，10000×２＝20000ヶの配列を用意する必要があるということです。

get 関数で読み込んだデータを画面上に表示する関数が，put 関数です。

【PUT】

書　式	put（x1，y1，x2，y2，na）
引　数	int　x1，y1，x2，y2 ｘ１……始点のｘ座標 ｙ１……始点のｙ座標 ｘ２……終点のｘ座標 ｙ２……終点のｙ座標 ｎａ……数値型の一次元配列名（パターンデータの入った配列名）

・x1，y1，x2，y2は window 関数で指定された範囲内の値

・配列内のデータ構造は get 関数と同じ

第３－55図のプログラムは，画面に二つの枠が指定され，左の枠の中でマウスの左ボタンを押すと水色の点が描かれますので，ドラッグしたりして適当な絵を描きます。次に，右ボタンを押すと右側の枠の中に今描いた絵がコピーされると同時に，"gamen.dat"というファイル名でディスクにセーブされ終了します。次に，**第３－56図**のプログラムを実行すると，ディスクから"gamen．dat"を読み込んで，画面に先程描いた絵を出力します。

```
 10 /* put & get test
 20 int mx,my,bl,br
 30 dim int gdata(12613)
 40 screen 1,2,1,1
 50 mouse(0)
 60 mouse(1)
 70 mouse(4)
 80 palet(1,hsv(96,31,31))
 90 palet(2,hsv(0,0,31))
100 box(0,100,250,300,2)
110 box(261,100,511,300,2)
120 msarea(0,100,250,300)
130 while br=0
```

```
140        msstat(mx,my,bl,br)
150        if bl=-1 then psetting()
160 endwhile
170 copy()
180 mouse(0)
190 end
200 func psetting()
210        int x,y
220        mspos(x,y)
230        pset(x,y,1)
240 endfunc
250 func copy()
260        int fn
270        get(0,100,250,300,gdata)
280        put(261,100,511,300,gdata)
290        fn=fopen("gamen.dat","c")
300        fwrite(gdata,12613,fn)
310        fclose(fn)
320 endfunc
```

第3－55図　put, get サンプルプログラム1

10：コメント文
20：変数宣言
30：配列宣言
40：画面モードを512×512，256色モードに設定
50：マウスの初期化
60：マウスカーソルの表示
70：右ボタンの開放
80～90：パレットコード1，2に水色，白色をセット
100：左側の白枠を描く。
110：右側の白枠を描く。
120：マウスカーソルの移動範囲を左側の白枠の中だけに設定する。
130：マウスの右ボタンが押されるまで，160行までの間をループする。
140：マウスのボタンが押されたかを検出する。
150：もし左ボタンが押されたならば，定義関数psetting（　）へ行く。

160：130行へ戻る。
170：右ボタンが押されたら，この行にきて定義関数copy（　）を呼び出す。
180：マウスの初期化
190：終了
200：定義関数psetting（　）の定義宣言
210：変数宣言（ローカル）
220：マウスカーソルの座標をx，yに読み込む。
230：読み込んだ座標x，yに水色のドットを描く。
240：定義関数終了宣言。呼ばれた次の行へ戻る。(160行)
250：定義関数copy（　）の定義宣言
260：変数宣言（ローカル）
270：左白枠内のパターンを配列gdataに読み込む。この場合，画面モードから１ドットは１バイトで表現でき，白枠内のドット総数は縦201×横251＝50451ドットになります。gdataは，int型の配列なので，一つの配列の中に４ドット分を格納できるので，50451／４≒12613ヶ分の配列をとればよいことになります。
280：右白枠内の位置に配列gdataに格納されているパターンを描く。
290：”gamen．dat”というファイルをオープンする。
300：配列gdataの値を”gamen．dat”というファイルに書き込む。
310：”gamen．dat”というファイルをクローズする。
320：定義関数終了宣言。呼ばれた次の行（180行）に戻る。

```
 10 /* put & get test 2
 20 int fn
 30 dim int gdata(12613)
 40 screen 1,2,1,1
 50 palet(1,hsv(96,31,31))
 60 palet(2,hsv(0,0,31))
 70 fn=fopen("gamen.dat","r")
 80 fread(gdata,12613,fn)
 90 fclose(fn)
100 put(0,100,250,300,gdata)
110 end
```

第３－56図　put & getサンプルプログラム２

10：コメント文
20：変数宣言

30：配列宣言

40：画面モードを512×512，256色モードに設定

50～60：パレットコード1，2に水色をセット

70：”gamen．dat”というファイルを読み込み専用でオープンする。

80：配列 gdata に”gamen．dat”というファイルの内容を書き込む。

90：”gamen．dat”というファイルをクローズする。

100：配列 gdata に読み込まれたパターンを画面に描く。

”graph．fnc”の最後の関数が，contrast 関数です。この関数は，画面の明るさをコントロールする関数です。

【CONTRAST】

書　式	contrast1（i）
引　数	int　i i＝0（暗い）～15（明るい）

・0～15までの16段階で，画面の明るさをコントロールできます。

第3－57図は，画面に円を描き，contrast 関数で徐々に画面を暗くし，何も見えなくなった状態で円の中を塗りつぶし，徐々に画面を明るくしていくプログラムです。

```
 10 /* contrast test
 20 int i,j
 30 screen 1,2,1,1
 40 palet(1,hsv(16,31,31))
 50 palet(2,hsv(96,31,31))
 60 circle(256,256,150,1,0,360,256)
 70 for i=0 to 15
 80    contrast(15-i)
 90    for j=0 to 500
100    next
110 next
120 paint(256,256,2)
130 for i=0 to 15
140     contrast(0+i)
150     for j=0 to 500
```

```
160      next
170 next
180 end
```

第３－57図　contrast サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを512×512，256色モードに設定
40～50：パレットコード１，２に黄色，水色をセット
60：画面の中央に半径150の黄色の円を描く
70～110：for～next 間をループ。ｉの値は０～15に変化する。この時，contrast 関数の引数は15－ｉで表されているので，15，14，13……０と変化する。90～100行の for～next ループは，空ループで，だんだん暗くするために時間を稼いでいるだけです。
120：円の中を水色で塗りつぶす。この処理は画面が暗いためわかりません。
130～140：70～110の処理の逆で，contrast 関数の引数を０，１，２，３……15と変化させ，だんだん画面を明るくしていく。

3-3-2　マウス関数

X68000にとってマウスは，ユーザーとの大切なインターフェースです。このため，X-BASIC にもマウスを効果的に利用するための関数群が用意されています。マウスを使用する場合は，グラフィックを使う時と同じようにマウスを使うための準備が必要です。この時，マウスの各初期設定を行う関数が「MOUSE」関数です。

【MOUSE】

書　式	m＝mouse（ｉ）
引　数	int　ｉ（ｉ…０～４） ｉ＝０　の時　マウスの初期化を行う。 ｉ＝１　の時　マウスカーソルを表示する。 ｉ＝２　の時　マウスカーソルを消す。 ｉ＝３　の時　マウスの状態を返す。 ｉ＝４　の時　ソフトキーボードの表示を禁止し，右ボタンを開放する。
戻り値	int　m 引数ｉが３以外の場合，通常０が返る。エラーの場合は，－１が返る。引数ｉが３

	の時は，マウスカーソルが表示されていれば－1が返り，表示していなければ0が返る。

マウスは，X-BASIC 起動時や mouse（0）によって初期化された時には，右ボタンを1度クリックするとマウスカーソルが表示され，もう一度クリックするとソフトキーボードが表示されるように設定されています。従って，右ボタンを使用するプログラムを作成する場合には，mouse（4）を実行して，ソフトキーボードが表示されないようにする必要があります。

マウスカーソルは，通常表示画面サイズの範囲内を移動することが出来ますが，ある範囲内だけを移動可能な状態にすることが出来ます。

このマウスの移動範囲を設定する関数が「MSAREA」関数です。

【MSAREA】

書　式	msarea（x1，y1，x2，y2）
引　数	int　x1，y1，x2，y2 x 1…移動範囲左上X座標 y 1…移動範囲左上Y座標 x 2…移動範囲右下X座標 y 2…移動範囲右下Y座標

移動範囲の最大値は，表示画面サイズとなります。

第3－58図は，実行させるとマウスカーソルが表示され，左上の座標（0，0）から左クリックしたところの座標に黄色の四角形を描き，その四角形の中がマウスの移動範囲となるプログラムです。

```
 10 /* msarea test
 20 int mx,my,bl,br,dx,dy
 30 screen 2,0,1,1
 40 mouse(0)
 50 mouse(1)
 60 mouse(4)
 70 palet(0,hsv(0,0,0))
 80 palet(1,hsv(32,31,31))
 90 while br=0
100        msstat(mx,my,bl,br)
110        if bl=-1 then mst()
```

```
120 endwhile
130 mouse(0)
140 end
150 func mst()
160     int x,y
170     mspos(x,y)
180     if x=0 then x=1
190     if y=0 then y=1
200     msarea(0,0,x,y)
210     box(0,0,dx,dy,0)
220     box(0,0,x,y,1)
230     dx=x:dy=y
240 endfunc
```

第3－58図　msarea サンプルプログラム

10：コメント文
20：変数宣言
30：画面モードを768×512の16色モードに設定
40：マウスの初期化
50：マウスカーソルを表示する
60：マウスの右ボタンの開放
70：パレットコード０に黒（透明）をセット
80：パレットコード１に黄色をセット
90：マウスの右ボタンが押されるまで120行までの間をループ
100：マウスのボタンが押されたかを見る
110：もし，左ボタンが押されたなら定義関数 mst（　）を呼び出す
120：90行へ戻る
130：マウスの右ボタンが押されるとこの行にきて，マウスを初期化する
140：終了
150：定義関数 mst（　）の定義宣言
160：変数宣言
170：マウスカーソルの座標を変数ｘ，ｙに読み込む
180：もし，Ｘ座標が０であれば変数ｘは，１とする
190：もし，Ｙ座標が０であれば変数ｙは，１とする
200：マウスの移動範囲を（０，０）から（ｘ，-ｙ）の範囲に設定する
210：前回の移動範囲を表す四角形を消す

220：現在設定されている移動範囲に黄色の四角形を描く
230：変数x，yの値を変数dx，dyに代入する
240：定義関数終了宣言　呼ばれた次の行に戻る（120行）

マウスが初期化されて，マウスカーソルを表示させた場合，マウスカーソルの位置は画面の左上に表示されます。マウスカーソルは，マウスを動かせばその移動量に応じて画面の中を自由自在に動きますが，マウスを動かさなくても「SETMSPOS」関数によって移動させることも出来ます。

【SETMSPOS】

書　式	setmspos（x，y）
引　数	int　x，y x…指定X座標 y…指定Y座標

x，yで指定された座標にマウスカーソルを表示します。

指定できる範囲は，msarea関数によって設定された範囲内です。

第3－59図は，setmspos関数によってマウスカーソルでサインカーブを描かせるプログラムです。

```
 10 /*setmspos test
 20 int x,y
 30 float ra
 40 screen 2,0,1,1
 50 mouse(0)
 60 mouse(1)
 70 palet(1,hsv(32,31,31))
 80 for x=0 to 720
 90     ra=x*(pi(1)/180)
100     y=sin(ra)*80
110     setmspos(x+5,y+256)
120     pset(x+5,y+256,1)
130 next
140 mouse(0)
150 end
```

第3－59図 setmsposサンプルプログラム

10：コメント文
20：変数宣言
30：変数宣言（float 型）
40：画面モードを768×512の16色モードに設定
50：マウスの初期化
60：マウスカーソルを表示する
70：パレットコード1に黄色をセット
80：0～720度までxを変化させる（for～next 文）
90：x度をラジアンに変換する
100：sin 関数でx度の正弦を求める
110：変数x，yで指定される座標にマウスカーソルを表示する
120：変数x，yで指定される座標に黄色のドットを描く
130：next 文
140：マウスの初期化

マウスは，ポインティングデバイスともいわれるように一種の座標入力装置であり，コンピュータ側は入力された座標データに基づいてさまざまな処理を行います。この座標データを読み込む関数が「mspos」関数です。

【MSPOS】

書　式	mspos（x，y）
引　数	int 型の変数（定数でないところに注意！）
戻り値	xという変数の中には，マウスカーソルの指すX座標が読み込まれる。 yという変数の中には，マウスカーソルの指すY座標が読み込まれる。

第3－60図は，マウスカーソルの指す位置のX，Y座標を画面に表示するプログラムです。マウスを使うプログラムを作成する場合，この関数は非常に重要ですのでしっかりと覚えておいて下さい。

```
10 /* mspos test
20 int x,y
30 mouse(0)
40 mouse(1)
50 while 1
60       mspos(x,y)
```

```
70        locate 5,5 : print "X=";x,"Y=";y
80 endwhile
90 end
```

第３－60図　mspos サンプルプログラム

10：コメント文
20：変数宣言
30：マウスの初期化
40：マウスカーソルの表示
50：80行までの間を無限ループ
60：マウスカーソルの位置のｘ，ｙ座標を変数ｘ，ｙに読み込む
70：変数ｘ，ｙの値を画面に表示する
80：50行に戻る
＊プログラムを終了する場合，break キーを押して下さい。

マウスを使ったプログラムを作成する場合に，もう一つ大切な関数があります。それは，マウスのボタンの状態を返してくれる「msstat」関数です。プログラムでは，普通この関数によって得たマウスボタンの状態でどのような処理を行うかを決定していきます。

【MSSTAT】

書　式	msstat（ｘ，ｙ，bl，br）
引　数	int 変数
戻り値	ｘ……Ｘ方向の相対移動量（－128～127） ｙ……Ｙ方向の相対移動量（－128～127） ｂｌ…左ボタンが押されていればー１，押されていなければ０ ｂｒ…右ボタンが押されていればー１，押されていなければ０

第３－61図は，どちらのボタンが押されたかを画面に出力するプログラムです。終了する時は，break キーを押して下さい。

```
10 /* msstat test
20 int x,y,bl,br
30 mouse(0)
40 mouse(4)
50 while 1
```

```
  60       msstat(x,y,bl,br)
  70       if bl=-1 then locate 10,5:print"left on! "
  80       if br=-1 then locate 10,5:print"right on!"
  90       if bl=0 and br=0 then locate 10,5:print"         "
 100 endwhile
 110 end
```

第3－61図 msstat サンプルプログラム

10：コメント文
20：変数宣言
30：マウスの初期化
40：マウスの右ボタンの開放
50：100行までの間の無限ループ
60：マウスのボタンの状態を変数 bl，br に読み込む
70：もし，左ボタンが押されていたら"left on！"と表示する
80：もし，右ボタンが押されていたら"right on！"と表示する
90：どちらも押されていなければ，空白を表示する
100：50行に戻る

マウスのボタンに関して，もう一つの関数が「msbtn」です。

この関数は，ボタンが指定の状態になるまでの時間を返す関数でダブルクリックを検出する時などに便利です。

【MSBTN】

書　式	bs＝msbtn（n，b，t）
引　数	int　n，b，t n…ボタンの状態 n＝0の時　ボタンが離されるまで n＝1の時　ボタンが押されるまで b…ボタンの左右 b＝0の時　左ボタン b＝1の時　右ボタン t…待時間の最大値 tの値が0，1，65535の場合は，ずっと待つ。
戻り値	int　bs

	マウスボタンがそれぞれの状態になるまでの時間を返します。 待時間の最大値を越えた場合は，－１を返します。 ドラッグされた場合は，０を返します。

第3－62図は，ダブルクリックを検出するプログラムですが，最初に待時間を聞いてきますので入力して下さい。ダブルクリックが成功すれば，ビープ音が鳴ります。

```
 10 /* msbtn test
 20 int x,y,bl,br,bs,t
 30 mouse(0)
 40 mouse(4)
 50 input"time = ";t
 60 while br=0
 70      msstat(x,y,bl,br)
 80      if bl=0 then continue
 90      bs=msbtn(0,0,t)
100      if bs<10 or bs>t-10 then continue
110      bs=msbtn(1,0,t)
120      if bs<10 or bs>t-10 then continue
130      bs=msbtn(0,0,t)
140      if bs<10 or bs>t-10 then continue
150      beep
160      print"Double click ok!"
170 endwhile
180 mouse(0)
190 end
```

第3－62図 msbtn サンプルプログラム

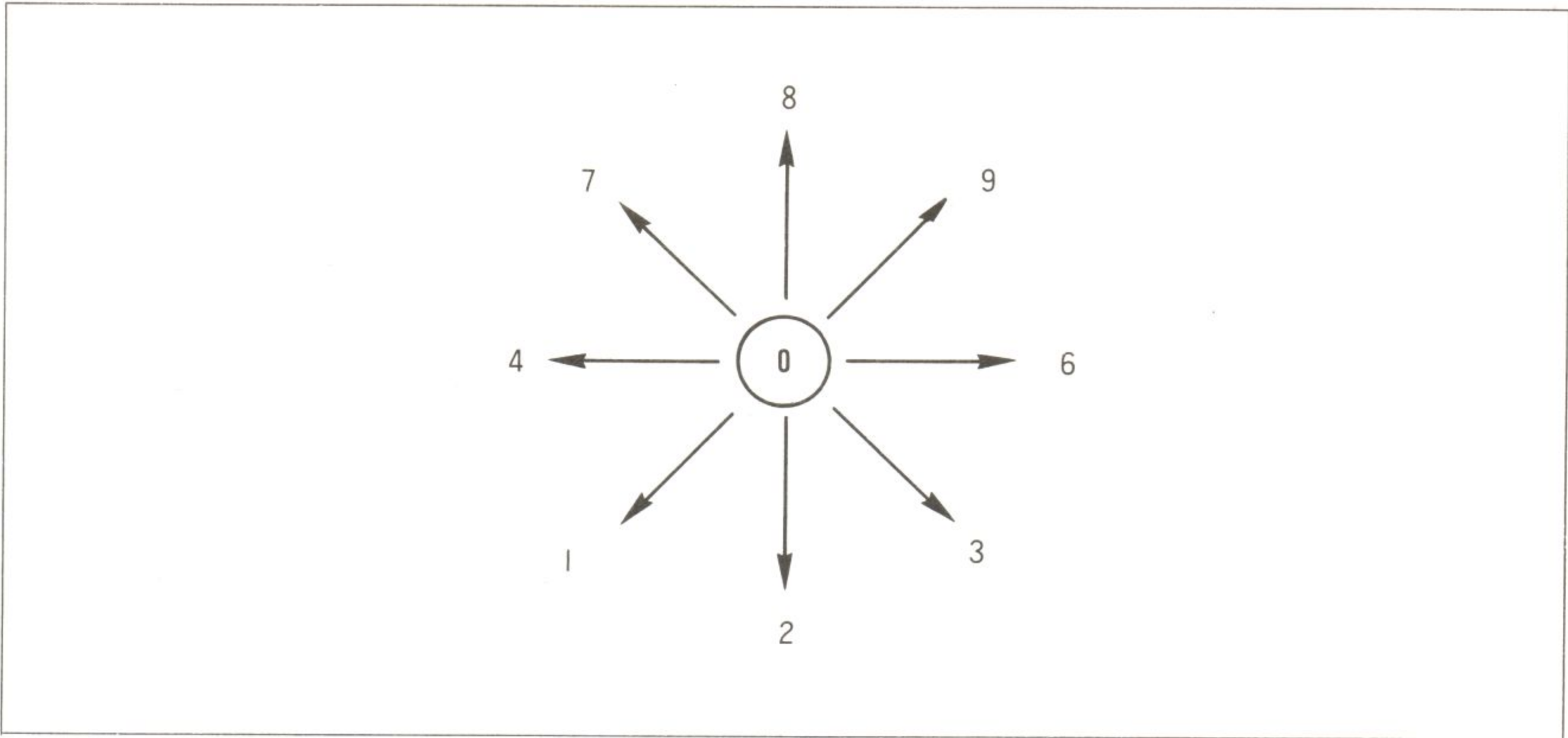
10：コメント文
20：変数宣言
30：マウスの初期化
40：右ボタンの開放
50：待時間の入力
60：マウスの右ボタンが押されるまで170行までの間をループ
70：マウスボタンの状態を見る

80：左ボタンが押されていなければ，60行に戻る
90：左ボタンが離されるまでの時間を変数 bs に返す。このとき待時間 t を越えたら－１を返す
100：bs の値が10より小さく，t－10より大きければ60行に戻る
110：左ボタンが押されるまでの時間を変数 bs に返す。このとき待時間 t を越えたら－１を返す
120：bs の値が10より小さく，t－10より大きければ60行に戻る
130：左ボタンが離されるまでの時間を変数 bs に返す。このとき待時間 t を越えたら－１を返す
140：bs の値が10より小さく，t－10より大きければ60行に戻る
150：ビープ音を鳴らす
160：ダブルクリックが成功したことを示すコメントを画面に出力する
170：60行に戻る
180：マウスを初期化

3-3-3　ジョイスティック関数

ここでは，ゲーム等を作る時に大切なジョイスティックに関する関数の説明を行います。X68000には素晴らしいゲームがこれからもたくさん出てくると思いますが，皆さんも一つ自分のゲームでも作って見たらいかがでしょうか？

【STICK】

書　式	sti＝stick（ i ）
引　数	int　i i ＝１のとき　ジョイスティック１のスティックの状態を調べる。 i ＝２のとき　ジョイスティック２のスティックの状態を調べる。
戻り値	int　sti スティックの倒れ方によって**第３－63図**のような値が返ってきます。

第3－63図　スティックの倒れ方と戻り値との関係

【STRIG】

書　式	trg=strig（i）
引　数	int　i i＝1のとき　ジョイスティック1のトリガーの状態を調べる。 i＝2のとき　ジョイスティック2のトリガーの状態を調べる。
戻り値	int trg trg＝0のとき　トリガー1，2とも押されていない。 trg＝1のとき　トリガー1のみ押されている。 trg＝2のとき　トリガー2のみ押されている。 trg＝3のとき　トリガー1，2とも押されている。

第3－64図は，ジョイスティックでマウスカーソルを動かして画面に線を描くプログラムです。トリガー1を押すと緑，トリガー2で水色の線となります。終了する時は，トリガー1，2を同時に押して下さい。

```
10 /* stic & strig test
20 int sti,trg,x,y,col=1
30 screen 2,0,1,1
40 mouse(0)
50 mouse(1)
```

```
 60 x=384 : y=256
 70 palet(1,hsv(64,31,31))
 80 palet(2,hsv(96,31,31))
 90 while trg<>3
100      sti=stick(1)
110      switch sti
120            case 1:x=x-1:y=y+1:pset(x,y,col):break
130            case 2:y=y+1:pset(x,y,col):break
140            case 3:x=x+1:y=y+1:pset(x,y,col):break
150            case 4:x=x-1:pset(x,y,col):break
160            case 6:x=x+1:pset(x,y,col):break
170            case 7:x=x-1:y=y-1:pset(x,y,col):break
180            case 8:y=y-1:pset(x,y,col):break
190            case 9:x=x+1:y=y-1:pset(x,y,col):break
200      endswitch
210      if x>767 then x=767
220      if y>511 then y=511
230      if x<0 then x=0
240      if y<0 then y=0
250      setmspos(x,y)
260      trg=strig(1)
270      switch trg
280            case 1:col=1:break
290            case 2:col=2:break
300      endswitch
310 endwhile
320 mouse(0)
330 end
```

第３－64図　ジョイスティック関数サンプルプログラム

10：コメント文

20：変数宣言

30：画面モードを768×512の16色モードに設定

40：マウスの初期化

50：マウスカーソルを表示
60：座標の初期値を変数x，yに代入
70～80：パレットコード1，2に緑，水色をセット
90：トリガー1，2がともに押されるまで310行までをループ
100：ジョイスティック1のスティックの状態を変数stiに返す
110～200：stiの内容によって点の表示座標を変えて点を描く
210～240：点の座標が，画面の外の値にならないように条件を与える
250：点の表示座標にマウスカーソルを表示する
260：ジョイスティック1のトリガーの状態を変数trgに返す
270～300：trgの内容によって描く色を変える
310：90行に戻る
320：マウスの初期化（トリガー1，2ともに押されていれば，ループを抜けて，この行に来る）

3-3-4 オーディオ関数

X68000では，ADPCMという方式によって音声をサンプリングすることが可能です。X68000の音声入力端子（AUDIO IN）から音声信号を取り込みデジタルデータに変換することによりメモリに格納することが出来るのです。

この素晴らしい機能もX－BASICから「audio.fnc」を使って利用することが可能となっています。

「audio.fnc」の中には，音声データをメモリに格納（録音）するための「A_REC」関数とメモリに格納されたデータを再生するための「A_PLAY」関数が用意されています。

【A_REC】

書式	a_rec（na，frq，le）
引数	na…pcmデータを格納する数値型の一次元配列の名前 int frq，le frq…サンプリング周波数 frq＝0の時　3.9KHz （1秒当たり1950バイトのメモリを消費） frq＝1の時　5.2KHz （1秒当たり2600バイトのメモリを消費） frq＝2の時　7.8KHz （1秒当たり3900バイトのメモリを消費）

	frq＝3の時　10.4KHz （1秒当たり5200バイトのメモリを消費） frq＝4の時　15.6KHz （1秒当たり7800バイトのメモリを消費） le…配列の添え字0からの長さ（バイト数） char 型配列の場合……最大値は，宣言した配列の値＋1 int 型配列の場合……最大値は，（宣言した配列の値＋1）×4 float 型配列の場合…最大値は，（宣言した配列の値＋1）×8

a_rec 関数は，pcm データを配列に格納します。従って，あらかじめ配列を用意しておく必要があります。この配列は数値型の一次元配列ですので，配列名を"pcm"とすれば，次の三つの型があります。

1．dim char pcm（n）　　n＝配列の数
2．dim int pcm（n）
3．dim float pcm（n）

char 型の場合には，一つの配列で1バイト，int 型の場合には一つの配列で4バイト，float 型では8バイトのデータが格納出来ます。

サンプリング周波数を最大の15.6KHz に設定した場合，1秒間の録音で7800バイトのメモリを消費するので，"dim char pcm（65535）"と配列宣言した場合には，65536バイト/7800バイト≒8秒間の録音が可能です。int 型で同じ配列を宣言すれば4倍の32秒間，float 型では8倍の64秒間の録音が可能です。しかし，"dim float pcm（65535）"という配列を宣言した場合，これだけで512K バイトのメモリが使われることになります。

引数leは，用意した配列の数の内，何個の配列（何バイト分）を使うかを決めます。例えば，"dim char pcm（65536）"という配列を用意して，"a_rec（pcm，4，32767）"とすれば4秒間の録音ということになります（32768バイト/7800バイト≒4）。"dim int pcm（65535）"とした場合は，配列の数は65536個ですが，バイト数は65536×4＝262144バイトとなり，le の最大値は262144となります。

次は，再生を行うための関数「A_PLAY」です。

【A_PLAY】

書　式	a_play（na，frq，md，le）
引　数	na……pcm データを格納する数値型の一次元配列の名前 int frq，md，le frq……サンプリング周波数 frq＝0の時　3.9KHz （1秒当たり1950バイトのメモリを消費） frq＝1の時　5.2KHz

(1秒当たり2600バイトのメモリを消費)
frq＝2の時　7.8KHz
(1秒当たり3900バイトのメモリを消費)
frq＝3の時　10.4KHz
(1秒当たり5200バイトのメモリを消費)
frq＝4の時　15.6KHz
(1秒当たり7800バイトのメモリを消費)
md……出力モード
md＝0の時　　出力しない
md＝1の時　　左チャンネルのみ出力
md＝2の時　　右チャンネルのみ出力
md＝3の時　　左右のチャンネルから出力
le……配列の添え字0からの長さ（バイト数）
char 型配列の場合……最大値は，宣言した配列の値＋1
int 型配列の場合………最大値は，（宣言した配列の値＋1）×4
float 型配列の場合……最大値は，（宣言した配列の値＋1）×8

第3－65図は，録音，再生を行うプログラムです。画面の指示にしたがって実行して下さい。もちろん，音声入力端子にはラジカセ等を接続しておく必要があります。

```
 10 /* a_rec & a_play test
 20 int frq,le,md
 30 str k
 40 dim char pcm(65535)
 50 input"録音周波数（０－４）";frq
 60 print"録音を始めます。何かキーを押して下さい。"
 70 k=inkey$
 80 a_rec(pcm,frq,65536)
 90 while 1
100     input"再生周波数（０－４）";frq
110     input"再生するバイト数（１－６５５３６）";le
120     print"再生を始めます。何かキーを押して下さい。"
130     k=inkey$
140     a_play(pcm,frq,3,le)
150     print"もう一度再生します。（Ｙ／Ｎ）"
```

```
160       k=inkey$
170       if k<>"y" and k<>"Y" then break
180 endwhile
190 end
```

第３－65図　オーディオ関数サンプルプログラム

10：コメント文
20～30：変数宣言
40：pcm データを格納する配列の宣言
50：録音周波数の入力
60：メッセージを画面に出力
70：何かキーが押されるまで待つ
80：配列 pcm に先程入力した周波数の pcm データを65536バイト分格納する
90：180行までの間を無限ループ
100：再生周波数を入力
110：再生するバイト数を入力
120：メッセージを画面に出力
130：何かキーが押されるまで待つ
140：配列 pcm の中の pcm データを先程入力した周波数で指定バイト数分を左右のチャンネルから出力する
150：メッセージを画面に出力
160：何かキーが押されるまで待つ
170：押されたキーが［Y］キー以外ならば190行へ飛ぶ
180：90行へ戻る

3-3-5 FM音源の制御(MUSIC.FNC)

X68000には，８重和音のステレオ FM 音源の LSI が内蔵されており，MUSIC.FNC として外部関数が用意されていて，このハードウエアを使用することが出来ます。しかし，この LSI があまりにも機能が豊富すぎて，使用に当たっては BASIC の関数としては不適切なほど内部をいじれるような関数もあります。

実際のところ，以前までの FM 音源の主流だった YM2203という LSI のマニュアルは，本のように分厚い物が付いてきました。逆を言えば，それほど FM 音源と言うのは面倒なものなのです。従ってここでは，一通り関数説明をして実際に曲を演奏する手順を解説していきます。

（a）FM音源の関数

【M_ALLOC】

トラックバッファを確保します。トラック内のデータは，クリアされますので注意して下さい。

書　式	mf=m_alloc（t， s）
引　数	char　t，int　s t……トラック番号（1～80） s……トラックサイズ（1～65536バイト）
戻り値	int　mf 戻り値としては，正常の時は0，エラーの時は－1を返します。

【M_ASSIGN】

FM音源の各チャンネルに，トラックの割り付けをします。複数のチャンネルに同一のトラックを割る当てると，それぞれ同じ楽譜を演奏します。また，複数のトラックを一つのチャンネルに割り当てると，一番最後に設定されたトラックの楽譜を演奏します。デフォルト値は，1から8チャンネルが1から8トラックに対応します。

書　式	mf=m_assign（c，t）
引　数	char　c，t c……チャンネル番号（1～8） t……トラック番号（1～80）
戻り値	int　mf 戻り値　mfは，正常の時は0，エラーの時は－1を返します。

【M_CONT】

c1からc8で指定されたチャンネルにおいて，一時停止した演奏を再開します。チャンネルを全て省略すると現在一時停止している全てのチャンネルの演奏を再開します。

書　式	mf=m_cont（c1，c2，c3，c4，c5，c6，c7，c8）

引　数	char
戻り値	int　mf 戻り値　mf は，正常の時は 0，エラーの時は－1 を返します。

【M_FREE】

指定されたトラックの残りバイト数を返します。

書　式	n＝m_free（t）
引　数	char　t t……トラック番号（1～80）
戻り値	int　n 戻り値　n は，トラックの残りバイト数を返します。

【M_INIT】

FM 音源（トラックデータ，音色）を初期化します。

書　式	m_init（　）
引　数	なし
戻り値	なし

【M_PLAY】

c1から c8で指定されたチャンネルに割り当てられたトラックデータを演奏します。チャンネルを全て省略すると，全チャンネルを演奏します。

書　式	mf＝m_play（c1，c2，c3，c4，c5，c6，c7，c8）
引　数	char
戻り値	int　mf 戻り値としては，正常の時は 0，エラーの時は－1 を返します。

【M_STAT】

指定されたチャンネルが演奏中かどうか調べ，演奏中なら1，そうでなければ0を返します。m_stat（　）の場合は，返されるバイト値のビット0～ビット7が，チャンネル1～8に対応します。

書　式	mf=m_stat（c）
引　数	char　c c……チャンネル番号（1～8）
戻り値	int　mf 戻り値として0なら停止中，1なら演奏中，エラーの時－1を返します。

【M_STOP】

c1からc8で指定されたチャンネルの演奏を，一時停止します。チャンネルを全て省略すると，全チャンネルの演奏を一時停止します。

書　式	mf=m_stop（c1，c2，c3，c4，c5，c6，c7，c8）
引　数	char
戻り値	int　mf 戻り値としては，正常の時は0，エラーの時は－1を返します。

【M_TEMPO】

テンポをセットします。

書　式	mf=m_tempo（te）
引　数	char　te te は，32～200の範囲です。
戻り値	int　mf 戻り値としては，正常の時は0，エラーの時は－1を返します。

【M_TRK】

MML による楽譜データをトラックに書き込みます。

書　式	mf=m_trk（t，st）
引　数	char　t，str　st t……トラック番号（1～80） s t…MML（ミュージックマクロランゲージ）データ
戻り値	int　mf 戻り値としては，正常の時は0，エラーの時は－1を返します。

m_trk 関数で使用する MML（Music Macro Language）には，その名の通り音楽的な文法を記号化した言語です。この記述は，文字列型変数に代入するか”（引用符）で囲んだ文字列を直接関数に記述するかのどちらかです。MML に使用される記号の文法は次の通りです。MML は，大文字でも小文字でも構いません。

◎　**音程に関する記号**

A～G…………A，B，C…は，ラ，シ，ド…ソ（ハ長調）に対応します。その後に#，または＋を付けると半音上がり，－を付けると半音下がります。

O0～O8…………オクターブの指定で，その後の0～8の数字が高さを表します。数字を省略すると，この前に設定したオクターブとみなされます。デフォルト値はO4です。なお，オクターブ0ではC，C#，Dは使えず，オクターブ8ではC，C#，D以外は使えません。

<………………オクターブを一つ上げます。

>………………オクターブを一つ下げます。

◎　**休符に関する記号**

R1～R64……休符を指定します。デフォルト値はR4です。

◎　**テンポに関する記号**

T32～T200…全体のテンポを設定します。デフォルト値は，T120です。

◎　**長さに関する記号**

L1～L64……音符の長さを決めます。L1を全音符（4拍）とし，以下の音の長さは1／数値となります。L64が最短で，64分音符になります。デフォルト値はL4です。

数字1～64……音符や休符の後に付け，Lの指定にかかわりなくその音の長さだけを変更します。音符や休符の長さを省略した場合は，その前のLの設定値になります。

．（ピリオド）…音符や休符の後に付け，その長さを1.5倍にします。二つ続けると，1.75倍になります。

Ｑ１～Ｑ８……１音中で，実際に音を出す割合を設定します。１～８の数字により，その割合は1/8～8/8（割合＝１）まで変化します。デフォルト値はQ8です。

&………………前後の音をつなぎます(タイ)。音色によっては，このコマンドを連続して使うと，音が小さくなります。

｛ ｝1～｛ ｝64…指定された長さのX分音符を，｛ ｝内の音色データの個数で等分した音の長さで出力します。分割した音の長さが，64分音符より短いとエラーになります。また，連符が増加（12連符など）すると，テンポに誤差が出る場合があります。また，｛ ｝内でL，Q等の音符の長さを変えるコマンドを使用すると，エラーとなります。

@L1～@L192…音の長さを短く指定します。全音符を192の長さとして，実際の長さを指定します。デフォルト値は，@L48（L4と同じ）です。

◎　**音量に関する記号**

V0～V15………音量を設定します。デフォルト値はV8です。

@V0～@V127…Vコマンドと関係なく音量を細かく設定します。

◎　**繰り返しに関する記号**

|:n，:|……|：nと：|の間をn回繰り返します。nは１～256で，省略値は２です。

| n……………繰り返しのn回目で，演奏すべき楽譜データを指定します。nは１～256です。

◎　**音色に関する記号**

@1～@200……音色を指定した番号のものにします。デフォルト値は@１です。音色番号は**第３－66図**に示す通りです。

◎　**その他の記号**

Ｙｒ，ｄ………OPMのレジスタを直接設定します。

r……………レジスタ番号

d……………データ

ただし，&H01，&H10，&H11，&H12，&H14のレジスタ番号は使えません。

@W1～@W64…指定された長さだけ，現在の状態を維持します。ただし，YコマンドでKEY ON/OFFした後でのみ有効です。省略値は，その前のLの値と同じになります。デフォルト値は，@W4です。

1. A.PIANO	2. H.PIANO	3. E.PIANO	4. CLAVINET
5. CELESTA	6. CEMBALO	7. A.GUITAR	8. E.GUITAR
9. W.BASS	10.E.BASS	11.BANJO	12.E.GUITAR
13.HARP	14.KOTO	15.P.ORGAN1	16.P.ORGAN2
17.E.ORGAN	18.ACCORDION	19.VIOLIN	20.CELLO
21.STRINGS1	22.STRINGS2	23.PIZZICATO	24.VOICE
25.CHORUS	26.GLASS HARP	27.WHISTLE	28.PICCOLO

29.FLUTE	30.OBOE	31.CLARINET	32.BASSOON
33.SAXPHONE	34.TRUMPET	35.HORN	36.TROMBONE
37.TUBA	38.BRASS1	39.BRASS2	40.HARMONICA
41.OCARINA	42.RECORDER	43.SAMBA WHISTLE	44.PANFLUTE
45.SNARE DRUM	46.RIMSHOT	47.BASSDRUM	48.TOM-TOM
49.TIMPANI	50.BONGO	51.TIMBALES	52.TRIANGLE
53.COWBEL	54.TUBLER BELL	55.STEEL DRUM	56.GLOCKEN
57.VIBRAPHONE	58.MARIMBA	59.H-H CLOSE	60.H-H OPEN
61.CYMBAL	62.SYN LEAD1	63.SYN LEAD2	64.AMBULANCE
65.STORM	66.LASER GUN	67.GAME SE1	68.GAME SE2

第3－66図　音色の種類

【M_VGET】

音色データを配列に読み込みます。戻り値としては，正常の時は0，エラーの時は－1を返します。

書　式	m_vget（vo，ca）
引　数	char（vo），char（4，10），配列名（ca）
戻り値	int

【M_VSET】

配列に設定されている音色データを，指定された音色番号のメモリバッファに登録します。戻り値としては，正常の時は0，エラーの時は－1を返します。

書　式	m_vset（vo，ca）
引　数	char（vo），char（4，10），配列名（ca）
戻り値	int

（b）FM音源関数を使ってみる

それでは，実際に今までに説明してきた関数を実際に使用して曲を演奏させてみます。**第3－67図**を打ち込んで下さい。

```
 10 str a[255]
 20 m_init()
 30 m_alloc(1,2000)
 40 m_assign(1.1)
 50 m_trk(1,"q7 @1 v10 o3 t100")
 90 a="cdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcr"
130 m_trk(1,a)
170 m_play()
180 end
```

第3－67図 何の曲？

このプログラムでは，まず最初にMMLの記述用の文字型変数aを255文字に確保して宣言しています。次に，FM音源の初期化をしています。次の，m_allocとm_assignはペアのようなもので，トラックバッファを確保し各チャンネルにトラックを割り付けます。次に演奏する曲の諸条件をトラック1に書き込みます。これは，90行と130行とで実行している音符データの書き込みと一緒に実行してもいいのですが，音符データとテンポ，楽器，オクターブ等の諸条件の設定とは分けた方がわかりやすいデータの記述になります。トラックバッファがいっぱいになるまでは次々にトラックバッファへ書き込むことが出来ます。そして，170行でめでたく曲の演奏ということになります。

”蛙の歌が……”と思わず口に出てきそうなピアノの演奏になったと思います。このプログラムは行番号が飛々になっていますが，次に色々と追加していくためですから，そのままの行番号で打ち込んで下さい。

今，実行したのは1音の演奏ですが，X68000に内蔵されているFM音源LSIは，8音まで同時に出すことが出来ます。蛙の歌が……とくれば輪唱ということになります。もう1音追加してみましょう。第3－67図のプログラムへ次の**第3－68図**のようにリストを追加して下さい。

```
 10 str a[255],b[255]
 20 m_init()
 30 m_alloc(1,2000):m_alloc(2,2000)
 40 m_assign(1.1):m_assign(2,2)
 50 m_trk(1,"q7 @1 v10 o3 t100")
 60 m_trk(2,"q7 @1 v10 o4 t100")
 90 a="cdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcrrrrrrrr"
100 b="rrrrrrrcdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcr"
130 m_trk(1,a)
140 m_trk(2,b)
```

```
170 m_play()
180 end
```

第3－68図　2重奏にする

このプログラムを実行して，感動の涙が出る方もあるかと思います。網掛けの部分を追加したのは，まずトラックバッファをもう一つ追加して，そのトラックをチャンネル2に割り付けました。これで，2チャンネル（2音）を使うことが出来るようになりました。あとは先程の要領で，文字列型変数bに音譜データを書き込みトラック2へ送ってやればいいのです。

もう少し凝ってみます。輪唱も3重にしてみたらどうでしょうか。第3－68図のプログラムに**第3－69図**のようにリストを追加して下さい。

```
 10 str a[255],b[255],c[255]
 20 m_init()
 30 m_alloc(1,2000):m_alloc(2,2000):m_alloc(3,2000)
 40 m_assign(1,1):m_aasign(2,2):m_assign(3,3)
 50 m_trk(1,"q7 @1 v10 o3 t100")
 60 m_trk(2,"q7 @1 v10 o4 t100")
 70 m_trk(3,"q7 @1 v10 o5 t100")
 90 a="cdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcrrrrrrrrrrrrredc"
100 b="rrrrrrrrcdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcrrrrredc"
110 c="rrrrrrrrrrrrrrrrcdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edc"
130 m_trk(1,a)
140 m_trk(2,b)
150 m_trk(3,c)
170 m_play()
180 end
```

第3－69図　輪唱

もう完璧な蛙の歌になってしまいました。楽器の弾けない人にとっては非常に心強い味方ですね。もうヤミツキになった気持ちで，第四音を追加してみましょう。第四音はベースパートにしてみます。第3－69図を**第3－70図**のようにして下さい。

```
10 str a[255],b[255],c[255],d[255]
20 m_init()
30 m_alloc(1,2000):m_alloc(2,2000):m_alloc(3,2000):m_alloc(4,2000)
40 m_assign(1,1):m_assign(2,2):m_assign(3,3):m_assign(4,4)
50 m_trk(1,"q7 @1 v10 o3 t100")
60 m_trk(2,"q7 @1 v10 o4 t100")
```

```
 70 m_trk(3,"q7 @1 v10 o5 t100")
 80 m_trk(4,"q7 @10 v5 o3 t100")
 90 a="cdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcrrrrrrrrrrrredc"
100 b="rrrrrrrcdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edcrrrrredc"
110 c="rrrrrrrrrrrrrrcdefedcrefgagfercrcrcrcrc8c8d8d8e8e8f8f8edc"
120 d="cdefedcrcdefedcrcdefedcrcdefedcrcdefedcrcdefedcr"
130 m_trk(1,a)
140 m_trk(2,b)
150 m_trk(3,c)
160 m_trk(4,d)
170 m_play()
180 end
```

第３－70図　ベースパーの追加

FM 音源関係の関数の説明をしてきましたが，基本的にはこれだけでかなり原音に近い音が出てしまいますから，夢中になって楽譜をコピーされる方も多いと思います。

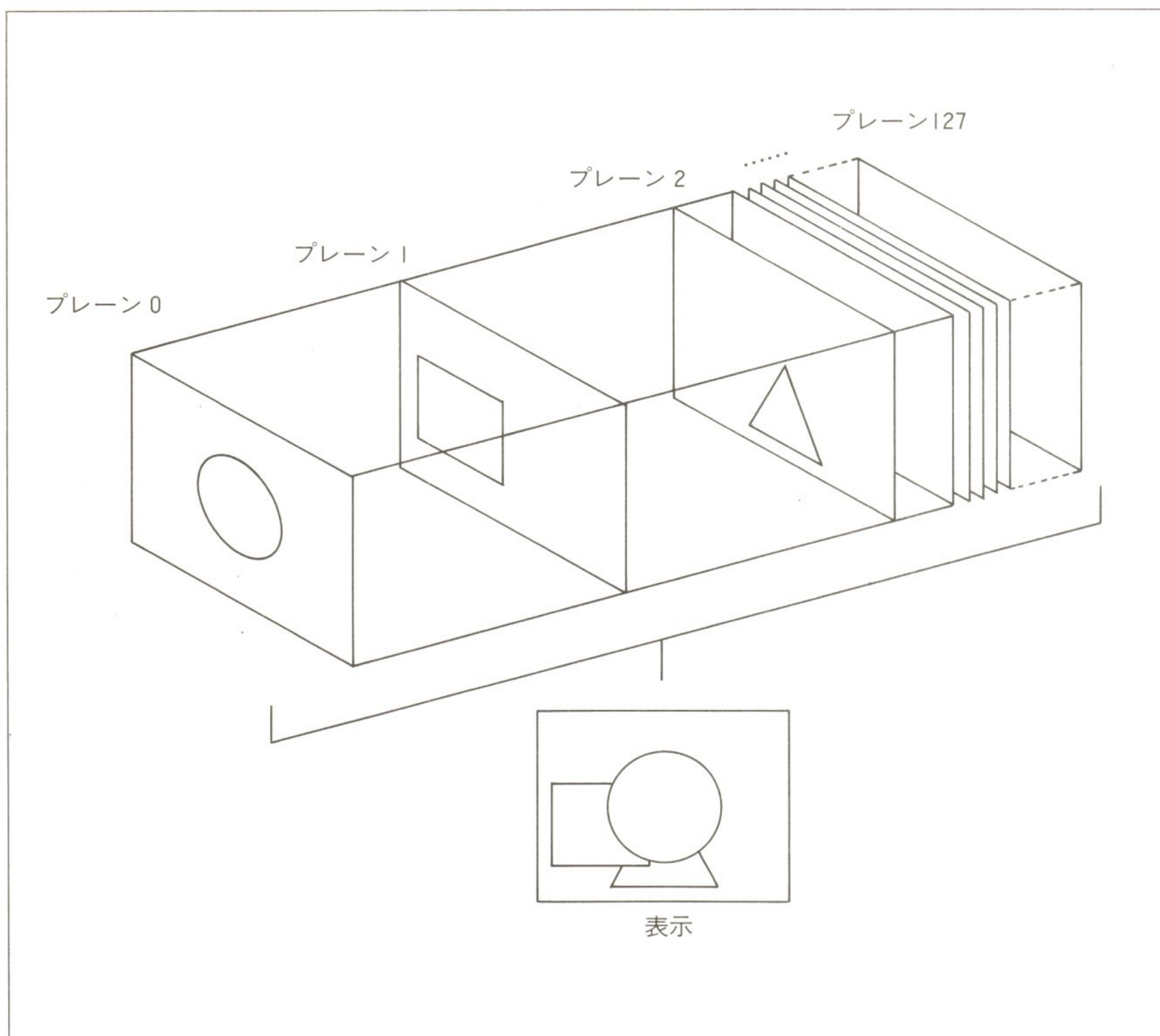
3-3-6　スプライト関数

X68000にはスプライト機能があり，自分で定義したキャラクタを高速かつスムーズに移動させることが可能です。このスプライト機能を X-BASIC 上で使うための関数群が「sprite. fnc」です。

●スプライト画面

テキスト，グラフィックに専用の画面があったのと同様に，スプライトパターンを表示する専用の画面がスプライト画面です。X68000にはスプライト画面が０から127までの128面あり，１枚につき一つのスプライトパターンを表示出来るので，画面上には最大で128個のスプライトパターンを表示出来ることになります。ただし，同じ水平位置には最大で32個しか表示出来ません。

128枚のスプライト画面には，０から127までの番号がついていて，各面をプレーン０，プレーン１，プレーン２……プレーン127と呼びます。複数のプレーンを表示する時には，プレーン番号の小さいプレーン上のパターンから手前に表示されます（**第３－71図**）。

また，テキスト，グラフィック画面との表示の優先順位は，X-BASIC では手前からスプライト・テキスト・グラフィック画面の順で表示されます。

第3－71図　　スプライト画面

●スプライトを使う

スプライト機能を使う時の手順は次のようになります。

1．画面モードの設定及びスプライト画面の初期化
2．スプライト画面の表示をONにする
3．スプライトパターン及びスプライトパレットの定義
4．使用プレーンの表示をONにする
5．スプライトパターンをプレーンに登録し，移動させる

この手順をもとに作成したプログラムが**第3－72図**でこのプログラムを見ながらスプライトについて説明していきます。

```
 10 /* sprite test1
 20 int x,y
 30 screen 1,3,1,1
 40 dim char pat(255)={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
 50                    1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,
 60                    1,2,3,3,3,3,3,3,3,3,3,3,3,3,2,1,
 70                    1,2,3,4,4,4,4,4,4,4,4,4,4,3,2,1,
 80                    1,2,3,4,5,5,5,5,5,5,5,5,4,3,2,1,
 90                    1,2,3,4,5,6,6,6,6,6,6,5,4,3,2,1,
100                    1,2,3,4,5,6,7,7,7,7,6,5,4,3,2,1,
110                    1,2,3,4,5,6,7,0,0,7,6,5,4,3,2,1,
120                    1,2,3,4,5,6,7,0,0,7,6,5,4,3,2,1,
130                    1,2,3,4,5,6,7,7,7,7,6,5,4,3,2,1,
140                    1,2,3,4,5,6,6,6,6,6,6,5,4,3,2,1,
150                    1,2,3,4,5,5,5,5,5,5,5,5,4,3,2,1,
160                    1,2,3,4,4,4,4,4,4,4,4,4,4,3,2,1,
170                    1,2,3,3,3,3,3,3,3,3,3,3,3,3,2,1,
180                    1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,
190                    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
200 sp_init()
210 sp_disp(1)
220 sp_def(1,pat,1)
230 sp_color(0,hsv(0,0,0),1)
240 sp_color(1,hsv(0,0,31),1)
250 sp_color(2,hsv(0,31,31),1)
260 sp_color(3,hsv(32,31,31),1)
270 sp_color(4,hsv(64,31,31),1)
280 sp_color(5,hsv(96,31,31),1)
290 sp_color(6,hsv(128,31,31),1)
300 sp_color(7,hsv(160,31,31),1)
310 sp_on(1)
320 for x=0 to 495
```

```
330      sp_move(1,x,256,1)
340 next
350 end
```

第３－72図　スプライト基本サンプルプログラム

スプライトは，表示画面サイズが768×512の時は使用出来ません。従って，表示画面サイズが512×512，または256×256になるように画面モードを設定する必要があるため，30行で画面モードの設定を行っています。この例では，512×512の65536色モードに設定しました。

次に200行でスプライト画面の初期化を行います。この初期化を行う関数が「SP_INIT」です。

【SP_INIT】

書　式	sp_init（ ）
引　数	なし

初期化が終わったら210行でスプライト画面の表示をONにします。

【SP_DISP】

書　式	sp_disp（ch）
引　数	char　ch ch＝０の時　　スプライト画面を表示しない。 ch＝１の時　　スプライト画面を表示する。

これで，スプライト画面の用意は完了です。いよいよスプライトパターンの定義に入ります。40行から190行までがパターンのデータです。一つのパターンは16×16ドットで表されて，そのデータはchar型の配列に格納することになっています。配列の数は16×16のデータを格納するのですから，256個用意することになります。１ドットのデータは，０から15で表されその値がスプライトのパレットコードになります。このパターンデータを定義する関数が220行の「sp_def」関数です。

【SP_DEF】

書　式	sp_def（pc，na，ps）
引　数	char　pc pc……パターンコード（０～255） 通常パターンは256個定義できるので，パターンにコードを付けて管理します。 この例の場合のパターンコードは１です。

	na……パターンデータを格納している配列名 　この例の場合は，"pat"です。 16×16ドットのパターンの場合の配列の数は256個，8×8ドットのパターンの場合は64個となります。 char　ps ps……パターンサイズ ps＝0の時　　8×8ドットのパターン ps＝1の時　　16×16ドットのパターン

8×8ドットのパターンは，現在説明しているスプライト画面では使用出来ません。このパターンは，後で説明する「バックグラウンド」で使用するものなので，普通はps＝1となります。

この例には出てきませんがスプライトパターンに関しては，あと二つ関数が用意されているのでついでに覚えておいて下さい。

【SP_CLR】

書　式	sp_clr（pcs，pce）
引　数	char　pcs，pce pcs……先頭パターンコード（0～255） pce……最終パターンコード（0～255） 指定範囲のパターンコードに登録されているパターンデータをクリアする。 引数が一つの場合は，そのパターンコードのデータだけをクリアし，引数がすべて省略された場合は0－255までのパターンコードのデータすべてがクリアされる。

【SP_PAT】

書　式	sp_pat（pc，na，ps）
引　数	char pc pc……パターンコード（0～255） na……パターンデータを格納する配列名 16×16ドットのパターンの場合の配列の数は256個，8×8ドットのパターンの場合の配列の数は64個となります。 char ps ps……パターンサイズ ps＝0の時　　8×8ドットのパターン ps＝1の時　　16×16ドットのパターン

この関数は，パターンコードに登録されているパターンデータを指定の配列の中に格納するものです。格納する順序は，スプライトパターンの左の一番上のドットデータを配列0に，右の一番上のドットデータを配列15（7）という順に配列255（63）まで格納していきます（（　）内は8×8ドットパターンの場合）。

ちょっと寄り道をしましたが，これでスプライトパターンの定義は完了です。次は，パターンの色を決めます。

グラフィックで256色モードの時は，パレットコードが0から255まであって，各パレットコードにカラーコードを設定して使いましたが，スプライトの場合はパレットブロックというものがあって一つのブロックの中の0から15までのパレットコードにカラーコードを設定します。パレットブロックは，15個用意されていますので65536色中240色をセットすることが出来ます。しかし，各パレットブロックのパレットコード0はカラーコード0（透明）にセットしておく必要があるので，実際には225色しか扱えません。スプライトパターンに色をセットする場合は，このパレットブロックで選択しますので，一つのスプライトパターンでは16色まで使えることになります。このパレットブロックに色をセットするのが「sp_color」関数です。

【SP_COLOR】

書　式	ocl=sp_color（pa，cl，pb）
引　数	char pa pa……パレットコード（0～15） int cl cl……カラーコード（0～65535） char pb pb……パレットブロックの番号（1～15）
戻り値	int ocl ocl……設定前のカラーコード

この例では，パレットブロック1のパレットコード0に透明，1に白，2に赤，3に黄色，4に緑，5に水色，6に青，7にマゼンダをhsv関数を使って設定しています（230行～300行）。

これでスプライトパターンのセットが完了したわけで，後は表示して動かすだけです。それにはまずどのプレーンを使用するか決めなくてはなりません。この例では，プレーン1を使用するため，プレーン1を表示ONにします。

【SP_ON】

書　式	sp_on（pls，ple）
引　数	char pls，ple

	pls……先頭プレーン番号（0～127） ple……最終プレーン番号（0～127） 指定範囲のプレーンの表示をONにする。 引数が一つの場合はその番号のプレーンだけ表示し，引数がすべて省略された場合は0－127番までのプレーンすべてが表示される。

310行では，1という引数だけなのでプレーン1だけ表示することになります。このプログラムでは使用していませんがsp_on関数の逆が「sp_off」関数です。

【SP_OFF】

書　式	sp_off（pls，ple）
引　数	char pls，ple pls……先頭プレーン番号（0～127） ple……最終プレーン番号（0～127） 指定範囲のプレーンの表示をOFFにする。 引数が一つの場合はその番号のプレーンだけOFFし，引数がすべて省略された場合は0－127番までのプレーンすべてが表示OFFとなる。

スプライトパターンを動かすための関数は，「SP_MOVE」関数です。この関数を使うことによって，スプライトパターンをプレーンに登録して表示することが出来ます。

【SP_MOVE】

書　式	sp_move（pl，x，y，pc）
引　数	int x，y x……表示位置Xの座標（－16～1007） y……表示位置Yの座標（－16～1007） スプライトパターンの左の一番上のドットの座標を表します。 char pl，pc pl……スプライトパターンを登録するプレーン番号を指定（0～127） pc……パターンコード（0～255）

この例では，パターンコード1にセットされているスプライトパターンをプレーン1に登録して表示Y座標を256に固定して，for～next文で表示X座標の値を変えて画面左から右まで移動させています（310行～330行）。

このsp_move関数は手軽に使える代わりに，パレットブロックは1に固定されているため，画面上で65536色中16色しか使えません。また，X68000のスプライト機能には，スプライトパターン

の上下左右の反転モードやバックグラウンドとの優先順位を決めることも出来ますが，このsp_move関数ではこれらの機能も使うことは出来ません。これらの機能を使うためには,「SP_SET」関数を使います。この関数は，sp_move 関数に比べると使い方が面倒ですが，実際にはこの関数があればsp_move 関数などいらないといった感じです。

【SP_SET】

<table>
<tr><td>書 式</td><td>sp_set (pl, x, y, cd, pr)</td></tr>
<tr><td>引 数</td><td>char pl
pl……プレーン番号（0～127)
int　x, y, cd
x……表示位置Xの座標（0～1023)
y……表示位置Yの座標（0～1023)
sp_move 関数の場合は，画面の一番左上が（16, 16）ですが，この場合は（0, 0）となります。
cd……スプライト設定コード（0～&hcfff)
スプライトの各設定を指定する。
bit 15　　0の時……通常表示
　　　　　1の時……上下反転表示
bit 14　　0の時……通常表示
　　　　　1の時……左右反転表示
bit13～bit12　使用しない（常に0）
bit11～bit 8　パレットブロックを表す
bit 7～bit 0　パターンコードを表す
char pr
スプライト画面とバックグラウンド画面との表示優先順位を決める。
pr＝0の時　スプライトは表示しない
pr＝1の時　手前から　BG0, BG1, SP
pr＝2の時　手前から　BG0, SP, BG1
pr＝3の時　手前から　SP, BG0, BG1
BG0……バックグラウンド第0面
BG1……バックグラウンド第1面
SP ……スプライト画面</td></tr>
</table>

ここで問題となるのは，cdの値の求め方だと思います。例えば，パターンコード2にセットしてあるスプライトパターンをパレットブロック8の色で上下を反転して表示させる場合の設定コードは次のようになります。

cd＝&B1000111100000010

=36610

第3－73図は sp_move 関数と sp_set 関数で，同じパターンを同じ座標に違うパレットブロックを使って表示させるプログラムです。

```
 10 /* sp_set test
 20 screen 1,3,1,1
 30 int x,y,v,h,pb,pc,cd
 40 dim char pat(255)={
 50      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 60      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 70      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 80      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 90      0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
100      0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,
110      0,0,0,0,0,0,0,1,3,1,0,0,0,0,0,0,
120      0,0,0,0,0,0,0,1,3,3,1,0,0,0,0,0,
130      1,1,1,1,1,1,1,1,3,4,3,1,0,0,0,0,
140      1,3,3,3,3,3,3,3,3,4,4,3,1,0,0,0,
150      1,3,4,4,4,4,4,4,4,4,4,4,3,1,0,0,
160      1,3,4,4,4,4,4,4,4,4,4,4,4,3,1,0,
170      1,3,3,3,3,3,3,3,3,3,3,3,3,3,3,1,
180      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
190      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
200      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
210   }
220 sp_init()
230 sp_disp(1)
240 sp_def(1,pat,1)
250 sp_def(2,pat,1)
260 for i=1 to 15
270      sp_color(i,hsv(0,0,31-2*i),1)
280      sp_color(i,hsv(0,31,31-2*i),2)
290 next
```

```
300 sp_color(0,hsv(0,0,0),1)
310 sp_color(0,hsv(0,0,0),2)
320 sp_on()
330 input"反転モード（上下）0 or 1 ";v
340 input"反転モード（左右）0 or 1 ";h
350 cd=sp_bit(v,h,2,1)
360 for x=0 to 495
370     sp_move(1,x,256,1)
380     sp_set(2,x,256,cd,3)
390 next
400 end
410 func sp_bit(v,h,pb,pc)
420      int bit
430      v=v*32768
440      h=h*16384
450      pb=pb*256
460      bit=v+h+pb+pc
470      return(bit)
480 endfunc
```

第3－73図　sp_move　，sp_set

10：コメント文
20：画面モードを512×512の65536色モードに設定
30：変数宣言
40～210：スプライトパターンデータを配列にセット
220：スプライトの初期化
230：スプライト画面表示ON
240：パターンコード1に配列内のスプライトパターンデータを定義
250：パターンコード2に配列内のスプライトパターンデータを定義
260～310：パレットブロック1と2にカラーコードを設定
320：すべてのプレーンの表示をONにする
330～340：スプライトパターンを反転表示させる場合は1を入力
340：定義関数sp_bit（v，h，pb，pc）を呼び出す
引数は，上下反転，左右反転，パレットブロック，パターンコードの順で入力するとcdに設定コードが返ってくる

360：X座標を0から495まで変える（390行までのループ）
370：パターンコード1をプレーン1に登録して座標（x，256）に表示
380：プレーン2に設定コードcdにより設定されたスプライトパターンを登録して，座標（x，256）に表示し，この時の優先順位SP＞BG0＞BG1の順とする
410～480：定義関数sp_bitの定義
この関数は与えられた引数により設定コードを計算して返します

このプログラムでは，同じ位置に二つの関数を使用してスプライトを表示していますが，関数の違いによって表示位置がずれることに注意して下さい。

この他に，スプライトの設定状態を返す「SP_STAT」関数があります。

【SP_STAT】

書　式	stat＝sp_stat（pl，md）
引　数	char pl，md pl……プレーン番号（0～127） md……読み出しデータの選択 md＝0の時　　Xポジション md＝1の時　　Yポジション md＝2の時　　設定コード md＝3の時　　画面優先順位（プライオリティ）
戻り値	int stat stat……スプライトの状態を返します。

●バックグラウンド

バックグラウンドとはスプライトの背景となる画面のことで，512×512の時は1面，256×256の時は2面表示することが可能で，1画面に64個×64個（計4096個）のパターンをセットすることが出来，各々の画面を自由にスクロールすることが出来ます。

バックグラウンドを使用する場合に注意することは，一つは画面モードを256×256に設定した場合にスプライトパターンが8×8の大きさになるので，パターンデータは8×8＝64個となります。従ってsp_def関数でパターンコードを設定する時には，3番目の引数を0にする必要があります。

もう一つは，sp_def関数で設定出来るパターンコードは0から255までの256個ですが，バックグラウンドを1面使うと0から191までの192個となり，2面とも使用すると0から127までの128個までしか定義出来ません。

これは，バックグラウンドに配置されるスプライトパターンのデータを記憶しておくために使われるためで，このエリアをテキストエリアと呼びます。テキストエリアはバックグラウンドを

2面使う場合は二つ，1面使う場合は一つ確保されます。

また，一つのテキストエリアには64個×64個のパターンを設定出来，パターンの位置を指定することが出来ます。

<512×512モード>

使用可能なバックグラウンド	No.0
使用するテキストエリア	No.0
バックグラウンド用パターン	16×16
スプライトパターン	16×16
パターン定義数	192個

<256×256モード>

使用可能なバックグラウンド	No.0，1
使用するテキストエリア	No.0，1
バックグラウンド用パターン	8×8
スプライトパターン	16×16
パターン定義数	1面のみ使用－192個 2面とも使用－128個

バックグラウンドを使用したサンプルプログラムが**第3－74図**です。このプログラムは，256×256モードでBG（バックグラウンド）を2面使用して各々違う方向へスクロールさせるものです。ここからは，このプログラムに基づいて実際にBGを使うための説明をしていきます。

```
10 /* back ground test
20 int x,y,i,v,h,pb,pc,cd,mx,my
30 /* sprite pattern data
40 dim char pat(255)={
50      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
 60      0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,0,
 70      0,9,9,9,9,9,9,9,9,9,9,9,9,9,10,0,
 80      0,10,9,8,8,8,8,8,8,8,8,8,8,9,10,0,
 90      0,10,9,8,7,7,7,7,7,7,7,7,8,9,10,0,
100      0,10,9,8,7,6,6,6,6,6,6,7,8,9,10,0,
110      0,10,9,8,7,6,5,5,5,5,6,7,8,9,10,0,
120      0,10,9,8,7,6,5,4,4,5,6,7,8,9,10,0,
130      0,10,9,8,7,6,5,4,4,5,6,7,8,9,10,0,
140      0,10,9,8,7,6,5,5,5,5,6,7,8,9,10,0,
150      0,10,9,8,7,6,6,6,6,6,6,7,8,9,10,0,
160      0,10,9,8,7,7,7,7,7,7,7,7,8,9,10,0,
170      0,10,9,8,8,8,8,8,8,8,8,8,8,9,10,0,
180      0,10,9,9,9,9,9,9,9,9,9,9,9,9,10,0,
190      0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,0,
200      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
210    }
220 /* BG1 pattern data
230 dim char bg1(64)={1,2,0,0,1,2,0,0,
240                   1,2,0,0,1,2,0,0,
250                   1,2,0,0,1,2,0,0,
260                   1,2,0,0,1,2,0,0,
270                   1,2,0,0,1,2,0,0,
280                   1,2,0,0,1,2,0,0,
290                   1,2,0,0,1,2,0,0,
300                   1,2,0,0,1,2,0,0}
310 /* BG0 pattern data
320 dim char bg(64)={0,0,0,0,0,0,0,0,
330                  0,0,0,0,0,0,0,0,
340                  0,0,0,0,0,0,0,0,
350                  0,0,0,1,1,0,0,0,
360                  0,0,0,1,1,0,0,0,
370                  0,0,0,0,0,0,0,0,
380                  0,0,0,0,0,0,0,0,
```

```
390                   0,0,0,0,0,0,0,0}
400   screen 0,3,1,1
410   sp_init()
420   sp_disp(1)
430   sp_on(1)
440   sp_def(1,pat,1)
450   sp_def(2,bg1,0)
460   sp_def(3,bg,0)
470 /* palet block set
480   for i=1 to 15
490       sp_color(i,hsv(64,31,20-i),1)
500   next
510   sp_color(0,0,1)
520   sp_color(0,0,2)
530   sp_color(1,hsv(128,20,30),2)
540   sp_color(2,hsv(128,20,20),2)
550   sp_color(0,0,3)
560   sp_color(1,hsv(0,0,20),3)
570 /* BG0 set
580   cd=sp_bit(0,0,3,3)
590   bg_fill(1,cd)
600   bg_set(0,1,1)
610 /* BG1 set
620   cd=sp_bit(0,0,2,2)
630   bg_fill(0,cd)
640   bg_set(1,0,1)
650 /* sprite set
660   cd=sp_bit(0,0,1,1)
670   sp_set(1,128,128,cd,3)
680 /* BG0 & BG1 scroll
690   while 1
700       if mx=511  then mx=0
710       if my=511  then my=0
```

```
720         mx=mx+1:my=my+1
730         bg_scroll(0,0,my)
740         bg_scroll(1,mx,0)
750         for i=0 to 100 : next
760     endwhile
770     end
780     func sp_bit(v,h,pb,pc)
790          int bit
800          v=v*32768
810          h=h*16384
820          pb=pb*256
830          bit=v+h+pb+pc
840          return(bit)
850 endfunc
```

第3－74図　バックグラウンドテスト

20：変数宣言
30～390：各パターンデータを配列に格納
　配列 pat　　普通のスプライトパターンデータ（16×16）
　配列 bg　　バックグラウンド No. 0 用のパターンデータ（8×8）
　配列 bg1　　バックグラウンド No. 1 用のパターンデータ（8×8）
400：画面モードを256×256にする
410：スプライトの初期化
420：スプライト画面表示 ON
430：プレーン1を表示 ON
440～460：各パターンコードにパターンデータを設定　3番目の引数に注意
480～560：各パレットブロックにカラーコードをセット
580：定義関数 sp_bit（v，h，pb，pc）を呼び出す
　引数は，上下反転，左右反転，パレットブロック，パターンコードの順で入力すると cd に設定コードが返ってくる
　この場合，反転モードは通常表示，パレットブロック3，パターンコード3
590：テキストエリアに設定コードで表されるデータを満たす

【BG_FILL】

書　式	bg_fill（tx，cd）

引　数	char tx tx……テキストエリア 512×512の時　　0 256×256の時　　0，1 int cd cd……スプライト設定コード （sp_set 関数参照）

この例では，580行で得た設定コード cd のデータでテキストエリア1を満たします。また，この他に一つずつのパターン位置を指定してテキストエリアにセットしていく「bg_put」関数があります。

【BG_PUT】

書　式	bg_put（tx，x，y，cd）
引　数	char tx，x，y tx……テキストエリア 512×512の時　　0 256×256の時　　0，1 x……テキストX座標（0～63） y……テキストY座標（0～63） int cd cd……スプライト設定コード （sp_set 関数参照）

600：バックグラウンドにテキストエリアを割り当て，表示をONにする。

【BG_SET】

書　式	bg_set（bg，tx，i）
引　数	char bg，tx，i bg……バックグラウンド No.（0～1） tx……テキストエリア No.（0～1） i……表示モード 　0の時　　表示OFF 　1の時　　表示ON

この例では，バックグラウンド０に先程セットしたテキストエリア１のデータをセットし表示します。

620：定義関数 sp_bit（v， h，pb，pc）を呼び出す
引数は，上下反転，左右反転，パレットブロック，パターンコードの順で入力すると cd に設定コードが返ってくる
この場合，反転モードは通常表示，パレットブロック２，パターンコード２

630：620行で得た設定コード cd のデータでテキストエリア０を満たします。

640：バックグラウンド１に先程セットしたテキストエリア０のデータをセットし表示します

660：定義関数 sp_bit（v， h，pb，pc）を呼び出す
引数は，上下反転，左右反転，パレットブロック，パターンコードの順で入力すると cd に設定コードが返ってくる
この場合，反転モードは通常表示，パレットブロック１，パターンコード１

670：プレーン１に設定コード cd により設定されたスプライトパターンを登録して座標（128，128）に表示し，この時の優先順位 SP>BG0>BG1の順とする

690：760行までの間を無限ループ

700：変数 mx の値が511を越えたら mx を０にする

710：変数 my の値が511を越えたら my を０にする

720：BG のスクロール座標を１ずつ増やす
mx……スクロール X 座標
my……スクロール Y 座標

730～740：BG をスクロールさせる

【BG_SCROLL】

書　式	bg_scroll（bg， x， y）
引　数	char bg bg……バックグラウンド No.（０～１） int x， y x……スクロールX座標 y……スクロールY座標 各座標の範囲は，256×256の時　　0～511 　　　　　　　　512×512の時　　0～1023

バックグラウンドのサイズは，表示画面サイズが512×512の時は1024×1024で，256×256の時は512×512となります。

この例では，BG0を縦方向にスクロールさせ BG1を横方向にスクロールさせています。

750：スクロールが速すぎるので空ループで時間稼ぎ

760：690行へ戻る

以上が，バックグラウンドを使用するためのプログラムの流れです。

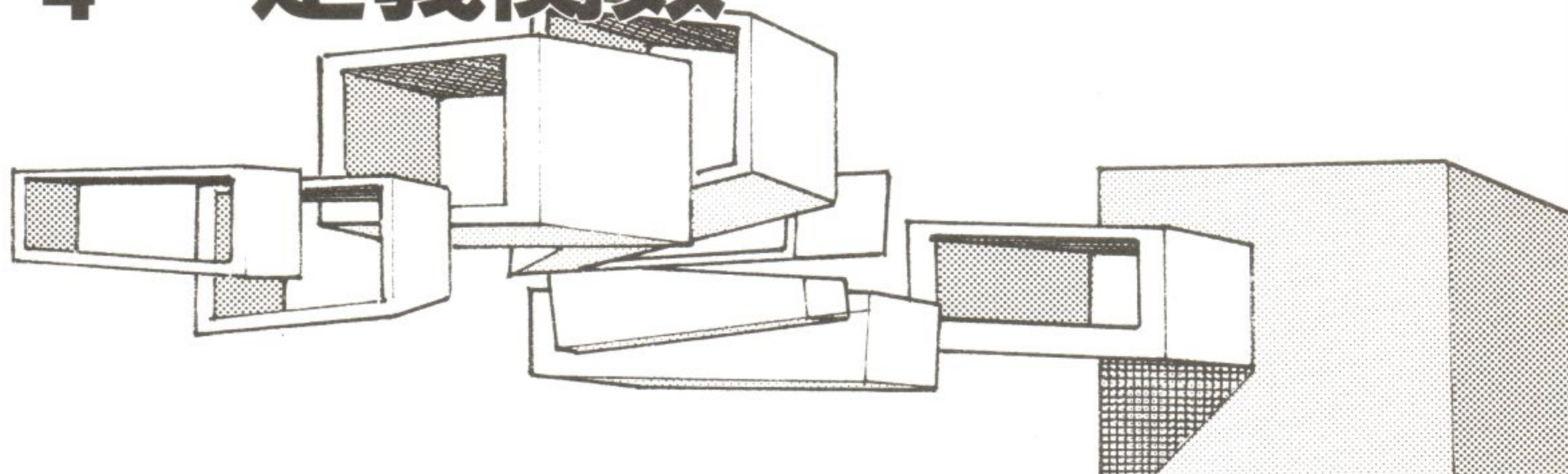
この他にバックグラウンドに関してあと二つの関数があります。

【BG_STAT】

書　式	stat=bg_stat（bg，md）
引　数	char bg，md bg……BG　No.（0～1） md……読み出てデータの選択 md＝0の時　　X座標 md＝1の時　　Y座標 md＝2の時　　テキストエリア md＝3の時　　表示のON/OFF
戻り値	int stat stat……バックグラウンドの状態を返します。

【BG_GET】

書　式	td=bg_get（tx，x，y）
引　数	char tx，x，y tx……テキストエリア 512×512の時　　0 256×256の時　　0，1 x……テキストX座標（0～63） y……テキストY座標（0～63）
戻り値	int td td……引数で指定された値のテキストデータを返します。

3-4　定義関数

X-BASICでは，従来のBASICでのステートメント（命令など）を関数化したことを特長としていますが，従来のBASICでは繰り返し実行される処理をサブルーチンとしてプログラムの簡略化とメモリ効率向上を実現しています。

X-BASICでは，従来のBASICでのサブルーチンについても関数の形で実現することが出来ます。定義関数については第2章で説明されていますが，ここでは2通りの定義関数の使い方について，実際にプログラム中でどのように使用されているかをサンプルプログラムによって説明します。

3-4-1　引数，戻り値を持たない関数

第3－75図は，マウスを使ってグラフィック画面に色々な図形を描くことが出来るプログラムです。このプログラムには，二つの関数が使われています。一つはdimwrite，もう一つはgwriteです。このプログラムに使用されている関数に共通していえることは，どちらの関数にも引数や戻り値がないということです。これは，前述の従来のBASICで使われていたサブルーチンと同じような使い方です。

X-BASICには，ローカル変数とグローバル変数の2種類の変数が存在しています。一方，従来のBASICにはグローバル変数だけしかありません。従って，関数へ処理が移る時に発生する引数の考え方やその結果として返ってくる戻り値などもありません。サブルーチンでは，メインプログラム中で使用されている変数に数値を入れて，その変数と同じ変数でサブルーチンで処理をしています。結果的にはこれが引数となるのですが，変数名を何時でも何処でも覚えていなければなりませんし，引数として渡した変数の内容がメチャメチャになって返ってくるなどの被害もたまにあります。

しかしながら，なかなか従来のBASICに慣れ親しんでいる人には，この感覚は忘れられないでしょう。そういった時に，このプログラムのような関数の使い方が便利です。

```
10 screen 2,0,1,1
20 int x,y,mx,my,bl,br,count
```

```
 30 dim int tate(10000),yoko(10000)
 40 mouse(0)
 50 mouse(1)
 60 mouse(4)
 70 msarea(0,0,767,511)
 80 palet(1,hsv(32,31,31))
 90 while 1
100     msstat(x,y,bl,br)
110     if bl=-1 then dimwrite()
120     if br=-1 then gwrite()
130 endwhile
140 end
200 func dimwrite()
210     mspos(mx,my)
220     count=count+1
230     yoko(count)=mx
240     tate(count)=my
250 endfunc
260 /*
270 func gwrite()
280     int i
290     if count=0 then count=1
300     for i=1 to count-1
310         line(yoko(i),tate(i).yoko(i+1),tate(i+1),1)
320     next
330     count=0
340 endfunc
```

第３－75図　引数，戻り値のない定義関数

さてこのプログラムは，グローバル変数として int 型で，x，y，mx，my，bl，br，count，同じくグローバル変数で，int 型の配列 tate（10000），yoko（10000）を宣言しています。この中で，int 型の x，y，mx，my，bl，br はマウス関数用に宣言したもので，プログラム中にマウスを使っている所で出て来ます。プログラムは，90行から130行がメインプログラムとなり，その前はマウスの使用やその移動範囲や画面の設定等の初期化部分です。メインプログラムは，while に

よる無限ループで，同じ処理が無限に繰り返されます。処理の中身はマウスの左ボタンが押されたらdimwriteを，右ボタンが押されたらgwriteを実行するだけのものです。あまりにも簡単なメインプログラムですが，メインプログラムの解読がこれほど簡単なこともX-BASICの特長です。

それでは関数の中へ入ってみます。

dimwrite（ ）……200行から260行までがdimwrite（ ）の関数です。この関数はマウスカーソルのある絶対位置mx，myを読み込んで，それぞれ縦，横の座標を入れておく配列tate( )，yoko（ ）へ配列の要素番号のcountを一つずつカウントアップして，記憶させていくものです。

gwrite（ ）……270行から340行までがgwrite（ ）関数です。この関数は，今まで蓄えられていたdimwrite（ ）関数での座標tate（ ），yoko（ ）を吐き出させる関数です。ただ吐き出させるのでは面白くないので，最初の座標tate（0）とyoko（0）からtate（1）とyoko（1）までを線で結び，次にtate（1）とyoko（1）からtate（2）とyoko（2）までを線で結び……といった具合いにcountまで繰り返させています。

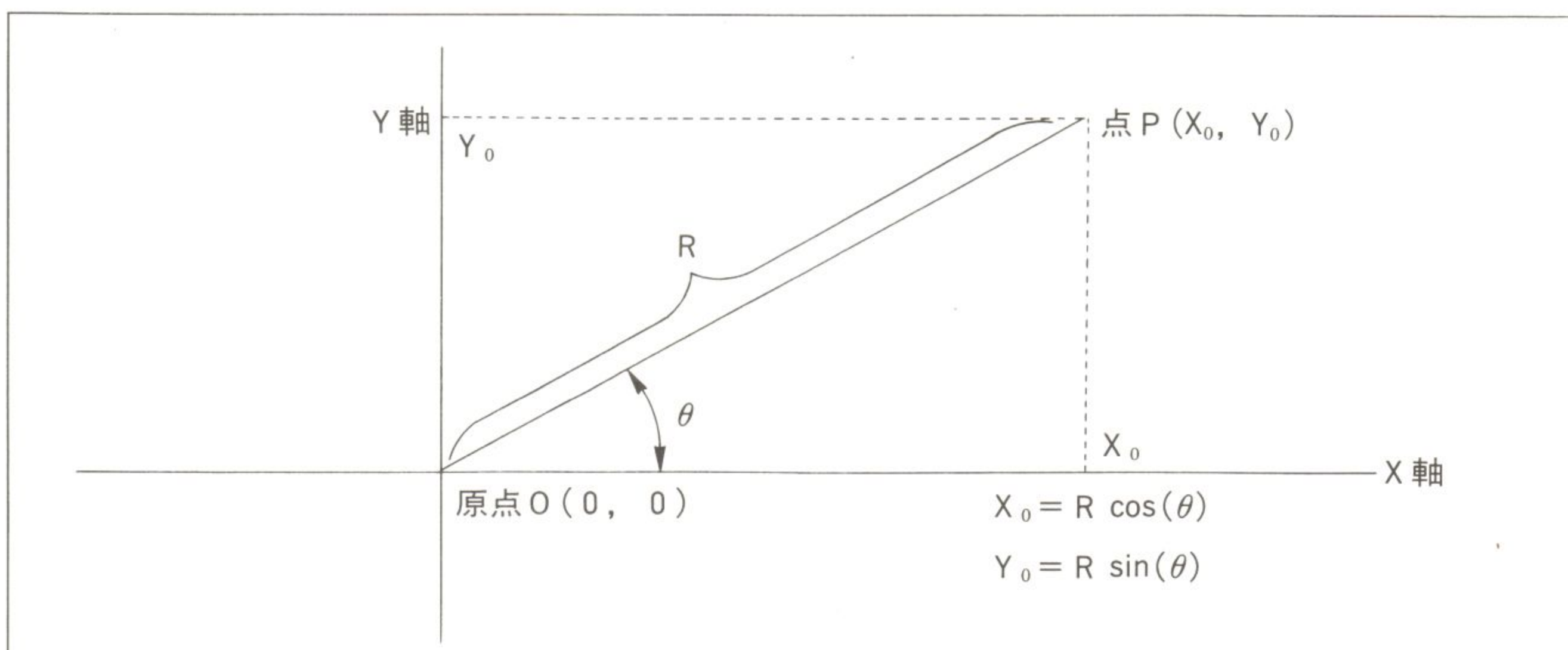
この二つの関数の全貌が明らかになった所で，プログラムの詳細が見えてきた方もいると思います。

3-4-2　引数，戻り値がある定義関数

3－4－1では，引数，戻り値がない関数の使用例を紹介しましたが，ここでは引数も戻り値もあります。GRAPH．FNCを思い出してみて下さい。色々な図形を描く外部関数があります。しかし，三角形を描く関数はありませんでした。ここでは，外部関数になかった関数を作り応用したサンプルプログラムを紹介します（**第3－77図**）。

本題に入る前に，このプログラムで使った数学的な手法について簡単に説明します。

原点O（0，0）からRだけはなれている点P（X_0，Y_0）があったとします。それぞれの点を結んだ線分O－P（長さR）とX軸との角度がθだったとします。この時の点P（X_0，Y_0）は，**第3－76図**の様に表すことが出来ます。この方法を極座標法といいグラフィックスで図形を描く時の座標計算によく使われます。

Y軸
点P（X_0，Y_0）
Y_0
R
θ
X_0
X軸
原点O（0，0）
$X_0 = R\cos(\theta)$
$Y_0 = R\sin(\theta)$

第3－76図　極座標法

さて，このプログラムはtriangle ( ) という関数一つしかありません。引数は，三角形を構成する3点とその線の色の情報です。戻り値は，色の情報が返ってくるだけで，そんなに大きな意味を持ちません。数値関数のようなものは，戻り値に命がかかっているのですが，このようなグラフィック等を扱う場合は，戻り値と同等の意味を持つのが実行結果，即ちディスプレイ上にあらわれる図形なのです。

プログラムは，先程の極座標法で求められる正三角形の3点の座標をforループで計算しながらtriangle ( ) を実行しています。

triangle( )……この関数は引数を七つ持っています。3点のx，y座標と線の色のパレットコードです。

```
 10 screen 2,0,1,1
 20 int ii,l=150
 30 float xx1,xx2,xx3,yy1,yy2,yy3,i
 40 for ii=0 to 120
 50      i=(pi(1)/60)*ii
 60      xx1=l*sin(i)+384:yy1=l*cos(i)+256
 70      xx2=l*sin(pi(2)/3+i)+384:yy2=l*cos(pi(2)/3+i)+256
 80      xx3=l*sin(pi(4)/3+i)+384:yy3=l*cos(pi(4)/3+i)+256
 90      triangle(xx1,yy1,xx2,yy2,xx3,yy3,5)
100 next
110 end
120 func triangle(a;float,b;float,c;float,d;float,e;float,f;float.cc)
130      line(a,b,c,d,cc)
140      line(c,d,e,f,cc+2)
150      line(e,f,a,b,cc+4)
160 endfunc
```

第3－77図　引数，戻り値のあるプログラム

3-4-3　その他の定義関数

このように2種類の定義関数がありましたが，この他にもX-BASICの特長を生かした関数の使用方法があります。再帰呼出しというものです。この方法は，グローバル変数とローカル変数があるため使用出来ます。ある関数が，その関数自身を関数内で使用しているというややこしいものです。パズルのようなものなので，さっと説明しますから悩んでみて下さい。**第3－78図**のプログラムが，再帰呼出しを使用したものです。

```
 10 int x
 20 input "x=",x
 30 print x;"!=";f(x)
 40 end
100 func int f(a;int)
110 if a=1 then return(1)
120 return(a*f(a-1))
130 endfunc
```

第3－78図　階乗を求めるプログラム

順を追って説明します。10行でint型変数が定義され，20行でその変数にキーボードから数値を代入します。次に，30行でその数値を引数とした定義関数f( )が，print文の書式で表示されプログラムを終了します。とても簡単なプログラムです。しかし，100行からの4行の関数の中には非常に大きな空間が広がっています。

例えば，xに5が代入されているとします。f(5)は110行のif文ではひっかからずに120行に行きます。120行のreturn( )の括弧の中に引数と関数自身に先程の引数から1引いたものを引数として掛けたものがあります。ここでこの関数は，自分自身をf(4)として呼び出します。再帰の始まりです。

f(4)は110行目にif文にひっかからずに，120行に行きます。120行のreturn( )の括弧の中に引数と関数自身に先程の引数から1引いたものを引数として掛けたものがあります。ここでこの関数は，自分自身をf(3)として呼び出します。

f(3)は110行目にif文にひっかからずに，120行に行きます。120行のreturn( )の括弧の中に引数と関数自身に先程の引数から1引いたものを引数として掛けたものがあります。ここでこの関数は，自分自身をf(2)として呼び出します。

f(2)は110行目にif文にひっかからずに，120行に行きます。120行のreturn( )の括弧の中に引数と関数自身に先程の引数から1引いたものを掛けたものがあります。ここでこの関数は，自分自身をf(1)として呼び出します。

f(1)は110行目にif文にひっかかりreturn(1)つまり戻り値が1となり前の関数に返ってきます。そこで前の関数f(2)での戻り値のreturn( )の括弧の中は，2＊1となります。これがf(3)に戻り，f(3)での戻り値のreturn( )の括弧の中は，3＊2＊1となります。これがf(4)に戻り，f(4)での戻り値のreturn( )の括弧の中は，4＊3＊2＊1となります。これがf(5)に戻り，f(5)での戻り値のreturn( )の括弧の中は，5＊4＊3＊2＊1つまり120という答えが30行のprint文で表示される訳です。

即ちこのプログラムは，最初に聞いてくる数値の階乗を求めるプログラムだったわけです。

ワンポイントテクニック　「BASIC. CNF」について

「BASIC.CNF」は，X-BASICの起動時の各種のモード設定や外部関数の組み込みに関する情報が書かれています。ここでちょっとこの「BASIC.CNF」の中身をのぞいてみることにしましょう。まず，ビジュアルシェルに戻ってみて下さい。BASICのウィンドウの中に「BASIC.CNF」というアイコンがありますので，アイコンメンテナンスで実行ファイルのところに〝ED.X〞と入力して登録します。これで「BASIC.CNF」のアイコンをダブルクリックすると〝ED.X〞（フルスクリーンエディタ）が起動されて「BASIC.CNF」の中身が画面に下記のように表示されます。

実行ファイルに〝ED.X〞を書き込む。

```
FREE   =128 ………………………… フリーエリアの設定
WIDTH  =64 ………………………… 画面横の文字数（64または96）
CAPS   =ON ………………………… ONで大文字，OFFで小文字
FUNC   =AUDIO  ……
FUNC   =GRAPH
FUNC   =MUSIC      …… 組み込む外部関数のファイル名（＊＊＊．FNC）
FUNC   =MOUSE
FUNC   =SPRITE
FUNC   =STICK  ……
```

例えば，起動時にフリーエリアを320Kバイト確保して，画面横の文字数は，96文字で小文字を使いたい場合は次のように内容を書き替えます。

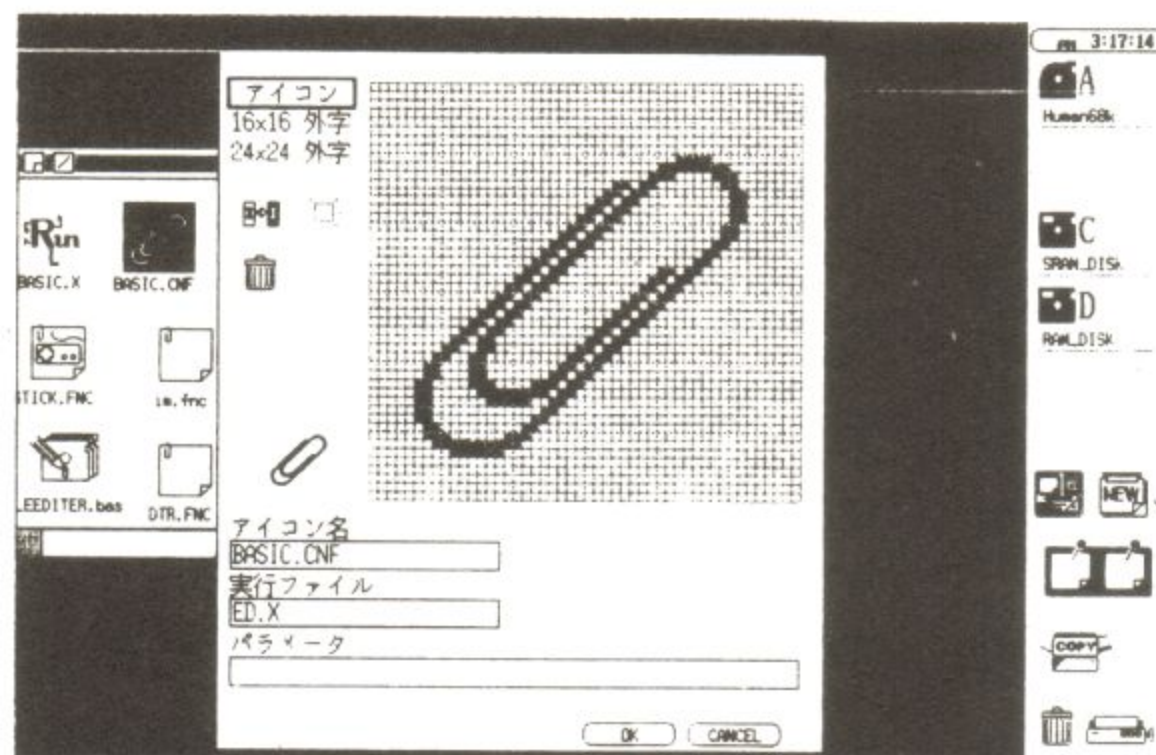
```
FREE＝320
WIDTH＝96
CAPS＝OFF
  :
```

また，「私はスプライトとFM音源は，使わないよ」という方は必要のない外部関数を削ることも出来ます。もちろん，組み込まれていない関数を使おうとすればエラーになります。

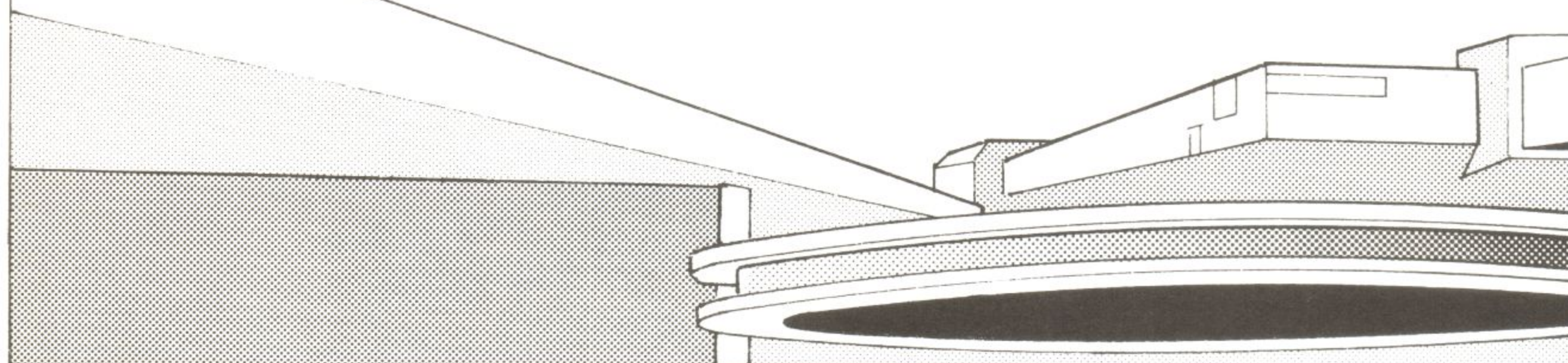
```
FREE＝320
WIDTH＝96
CAPS＝OFF
FUNC＝AUDIO
FUNC＝GRAPH
FUNC＝MOUSE
FUNC＝STICK
```

また，新しい外部関数を組み込むことも出来ます。例えば，イメージユニットを買った方は，福袋の中の「IMAGE.FNC」というファイルをこのBASICのディレクトリの中に移動してから下線の部分を追加すればX-BASICでイメージユニットが制御出来るようになります。

```
FREE＝320
WIDTH＝96
CAPS＝OFF
FUNC＝AUDIO
FUNC＝GRAPH
FUNC＝MOUSE
FUNC＝STICK
FUNC＝IMAGE……………………………………… 「IMAGE.FNC」の追加
```

変更が終わったら［ESC］キーを押した後で［E］キーを押せば「BASIC.CNF」が書き替えられます。
ほんのちょっとしたことですが，この「BASIC.CNF」があるおかげでX-BASICは非常に自由度が高いものになっています。これから発売されるであろう立体スコープや各種の拡張機器にもきっとその機能に応じた外部関数がサポートされて「BASIC.CNF」をちょっと書き替えるだけでX-BASICから簡単に制御出来るようになるでしょう。楽しみですね！

第4章

外部定義関数

4-1 外部定義関数の書式(その1)

X-BASICの標準関数，外部関数，定義関数について第3章で説明してきましたが，ここでは X-BASICの特長としてあげられる外部定義関数について説明します。これまでに標準関数，外部関数，そして各種ステートメントや演算等により独自の定義関数をプログラム内に持つことが出来ることは理解されたと思います。しかし，あくまでも BASIC で記述してありますから，BASIC の致命的な部分である実行速度が遅いことや，限られたステートメントや標準関数，外部関数では X68000のハードウエアの細かな部分までは操作することが出来ません。このような場合にこの外部定義関数を使用するわけです。しかしながら，今までのように何か基本となるもの（ステートメントや関数群）があり，それを組み立ててきたようなわけにいきません。外部定義関数は，68000のマシン語により作られているためです。

マシン語により作成される外部定義関数といっても関数ですから，前述のように引数や戻り値があります。またこの引数，戻り値についても四つの型があります。従来の BASIC では，X-BASIC のような外部定義関数はないので，マシン語を使用して外部にサブルーチンを置いて使用していました。この場合，引数や戻り値はなくもっぱらメモリ上に情報を書き込んではCALL するといった方法でした。X-BASIC の場合にはそんな無秩序なものではなく，ある決まった書式（フォーマット）によって外部関数を定義しています。基本的な書式は**第4-1図**のようになります。

```
*************** インフォメーション・テーブル*************************

adr_init        dc.l        return
adr_run         dc.l        return
adr_end         dc.l        return
adr_system      dc.l        return
adr_break       dc.l        return
adr_input       dc.l        return

adr_reservel    dc.l        return
```

```
adr_reserve2    dc.l        return
adr_token       dc.l        token_table
adr_parameter   dc.l        param_adr_table
adr_exec        dc.l        exec_table
reserve         dc.l        0,0,0,0,0

*************** ﾄｰｸﾝ･ﾃｰﾌﾞﾙ*****************************

token_table     dc.b        "tline",0,0

*************** ﾊﾟﾗﾒｰﾀ･ﾃｰﾌﾞﾙ***************************
                .even
param_adr_table dc.l        param_table_1

param_table_1   dc.w        $0002,$0002,$0002
                dc.w        $0002,$0002,$8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ･ﾃｰﾌﾞﾙ***********************
                .even
exec_table      dc.l        tline

*************** FUNC MAIN *****************************
                .even
tline:・・・・・　ここからマシン語のプログラムが始まる。
```

第４－１図　外部定義関数の基本的な書式

第４－１図は，関数本体の内容が変わってもここは変わることがありません。この部分のことを**ヘッダ**といい関数の引数，戻り値，関数名について定義しているところです。このヘッダを大きく分けると，四つに分かれます。

（１）インフォメーションテーブル

（２）トークンテーブル

（３）パラメータテーブル

（４）実行アドレステーブル

この四つについて，それぞれ解説してみます。

（1）インフォメーションテーブル

上から8行は，X-BASICが起動された時やプログラム実行時，終了時，ブレーク時などにおいて，それぞれの場合の実行アドレスを記述する部分です。しかし，ここは第4-1図のようにreturnというラベルをふっておいても結構です。returnというのは，当然この関数を抜ける部分（マシン語の終了部分）に定義されていて68000のアセンブラ記述で，rtsとなっている必要があります。その下の行から（2），（3），（4）のそれぞれのテーブルが宣言されています。

（2）トークンテーブル

ここはX-BASICで今作ろうとしている関数を，どのような関数名にするかを記述する部分です。この例の場合では，tlineという関数になっています。そして関数名の後には後述する複数の関数を同時に定義する時に関数名どうしを区切る”0”を入れます。また，この後にトークンテーブル終了を知らせる”0”も入れます。従って，この例のように一つだけ関数を定義する時は，関数名，”0”，”0”となります。

（3）パラメータテーブル

ここでは，関数にとって一番大切な引数と戻り値を定義します。引数，戻り値にはX-BASICの変数同様，基本的には四つの型があります。また定義関数では扱えなかった配列を使用した引数が定義出来るようになっています。第4-1図では，＄0002，＄8001等と書いてある部分は，これらの引数や戻り値にパラメータIDを割り付けたもので，その詳細は**第4-2図**のようになります。

引数	float……$0001	
	int………$0002	引数をそのまま受け取る場合
	char………$0004	
	str………$0008	
	float……$0011	
	int………$0012	引数をポインタとして受け取る場合
	char………$0014	
	str………$0018	
	float……$0081	
	int………$0082	引数が省略出来る場合
	char………$0084	
	str………$0088	
戻り値	float……$8000	
	int………$8001	
	char………$8003	
	無し………$ffff	

第4-2図　パラメータID

この他に，配列を引数としたものに次のようなものがあります。

配列の引数　float …………$0031
　　　　　　int ……………$0032　1次元配列の引数のID
　　　　　　char …………$0034
　　　　　　str ……………$0038

このパラメータIDをパラメータテーブルに必要な個数設定します。ただし，戻り値は一つだけです。この例では，int型の引数を五つと，int型の戻り値が一つ設定されています。

（4）実行アドレステーブル

色々な条件（X-BASICとの接点）について設定が出来ましたが，まだマシン語をどこから実行していいか分かりません。ここでは，マシン語のプログラムがどこから始まるのかを実行したい部分に記述したラベルで知らせてやります。この例の場合は，関数名と同じtlineです。従って，第4－1図の最下行にあるtline：からプログラムを実行するわけです。

ワンポイントテクニック　X-BASICでの数値表現

皆さんが日常良く使っている数値は，10進法です。X-BASICでは，この10進法の他に2進法と8進法と16進法の数値をダイレクトに表現することが出来ます。それぞれの表現方法は次の通りです。

◎　**10進法**

１２３４　　０．１１２２

◎　**2進法**

&Ｂ１０１０１０……10進法で４２

&Ｂを付けることで，２進法と解釈されます。

◎　**8進法**

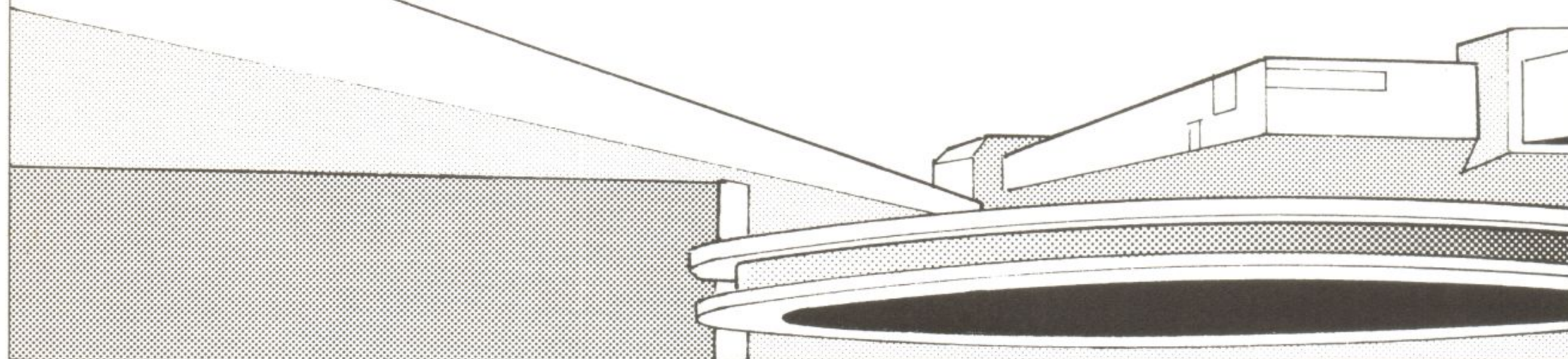
&Ｏ７２５……10進法で４６９

&Ｏを付けることで，８進法として解釈されます。

◎　**16進法**

&Ｈ７ＦＦ……10進法で２０４７

&Ｈを付けることで，16進法と解釈されます。

4-2　外部関数定義の実習

ここでは簡単な外部関数を実際に定義して解説していきます。まず最初に，色々な条件から決めなければなりません。

ここで紹介する方法は，一般的に仕事の打ち合わせの時の覚え書きや商品のスペックを説明してある仕様書のようなもので，関数の目的や，使い方，引き数，戻り値等を列記しておくものですが，こうしておくと後で色々と役に立つので自分なりの方法でまとめておくことをお薦めします。

関数名：oddeven（　）
引数の数と型：str 型で一つ
戻り値の型：int 型
関数の内容：引数で与えられた文字列が偶数であるか奇数であるかをチェックします。偶数であれば0，奇数であれば1を戻り値として返します。全角文字は2文字とし，文字列の終わりを示すヌルコードは数えません。

第4－3図　外部定義関数oddeven（　）の仕様

以上の仕様書で外部関数を定義してみます。

手順1

この仕様書で列記した条件をヘッダで使用出来るように加工します。引数と戻り値を先程のパラメータ IDの表で選びます。

引　数　　$0008
戻り値　　$8001
関数名（X－BASIC 側）
　　　　　oddeven(　)

となります。

手順2

ヘッダの作成をしますが，第4－1図をそのまま使い，引数と戻り値と関数名だけを書き換えます。このように変更する必要のない所をそのままにしてヘッダを作ると非常に能率がよく

なります。

```
**************** ｲﾝﾌｫﾒｰｼｮﾝ･ﾃｰﾌﾞﾙ************************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

**************** ﾄｰｸﾝ･ﾃｰﾌﾞﾙ******************************

token_table     dc.b            "oddeven",0,0

**************** ﾊﾟﾗﾒｰﾀ･ﾃｰﾌﾞﾙ***************************
                .even
param_adr_table dc.l            param_table_1

param_table_1   dc.w            $0008
                dc.w            $8001

**************** ｼﾞｯｺｳｱﾄﾞﾚｽ･ﾃｰﾌﾞﾙ************************
                .even
exec_table      dc.l            oddeven

**************** FUNC MAIN *******************************
                .even
oddeven:                                  網掛けの部分が変更したところです。
```

第4－4図　ヘッダ部分

手順3

oddeven：からプログラムを作ります。引数のIDや戻り値のIDはわかりましたが，実際のプログラム中でどのように渡されてくるかについて説明します。

このoddeven：のプログラム**第4－5図**の中に，

move.l　　12（sp），a0

という部分が先頭にあります。これは，文字列の格納されているメモリの先頭アドレスをa0に複写しています。引数はスタックポインタに渡されそこから，改めてレジスタに渡されます。str型の引数の場合には，自動的にポインタ（格納してあるメモリ番地）を渡す約束になっています。その他の数値の引数は，この例のように記述して，値をa0に複写します。

複数の引数がある時は，次のように2個目以降の引数が渡されます。

```
move.l   12(sp),xx…… 1番目の引数
move.l   22(sp),xx…… 2番目の引数
move.l   32(sp),xx…… 3番目の引数
move.l   42(sp),xx…… 4番目の引数
```

xxはプログラム内部で使用しているレジスタで，ユーザーが決める。ポインタの場合はaxで，値の場合はdxに渡した方が便利です。

```
oddeven:      move.l     12(sp),a0
              move.l     #-1,d0
count         addq.l     #1,d0
              tst.b      (a0)+
              bne        count
              and.l      #1,d0
              move.l     d0,ret_param+6
              clr.l      d0
              lea.l      ret_param(pc),a0
              lea.l      error_comment(pc),a1
return        rts

               even
ret_param     dc.w       0
              dc.l       0
              ds.l       1

error_comment dc.b       "error",0
```

```
                    end
```

第4－5図　oddeven（　）プログラム本体

さて，戻り値ですが，三つのレジスタにセットします。戻り値を格納したメモリのアドレスと，エラーの場合のメッセージを格納したメモリの先頭アドレスと，関数の終了時の状態を表すステータスです。

格納するレジスタは次の通りです。

d0：正常終了の場合は0

a0：戻り値を格納したメモリのアドレス

a1：エラーメッセージの格納してあるメモリの先頭アドレス

プログラム中で，ret_param としてある部分が戻り値を格納してある所です。さて，プログラムの説明ですが，前述の通りa0に文字列の先頭アドレスが移され，d0には－1がセットされました（1，2行目)。次にtst.b (a0)＋で，文字列が0がどうか（終わりかどうか）をチェックしながらループカウンタのd0をインクリメント（1加算）していきます。tst.b/ (a0)＋は実行後，a0を1加算しますから文字列の引数格納アドレスを1文字ずつ更新していきます。tst.b(a0)＋で0を検出した後，and.l #1，d0という命令で，最下位ビットだけをそのままにして残りの31ビット全てを0にします。こうしてしまえば，最後に残った最下位ビットが0なら偶数で，1なら奇数ということになります。

手順4

こうして机上でコーディングされたプログラムを，X68000付属のED.Xを使い入力していきます。入力する時には，"ファイルネーム".Sとしてアセンブルソースファイルであることを明示して下さい。この後，AS.XとLK.Xを使用してアセンブル及びリンクして下さい。

AS.Xを使用する時は，ファイルネームに＊＊＊.Sを指定，LK.Xでリンクする時は，ファイルネームに＊＊＊.Oを指定します。

手順5

出来上がった関数のファイルネームは，＊＊＊.Xとなっているのですが，X－BASICのディレクトリの中にある外部関数は全て＊＊＊.FNCとなっています。実は，外部関数は＊＊＊.Xだったのです。これは，前述のヘッダに秘密があり，ファイルネームを＊＊＊.Xから＊＊＊.FNCと変更するだけでよいのです。

```
*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ*******************************

adr_init        dc.l        return
adr_run         dc.l        return
adr_end         dc.l        return
```

```
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

*************** ﾄｰｸﾝ･ﾃｰﾌﾞﾙ*****************************

token_table     dc.b            "oddeven",0,0

*************** ﾊﾟﾗﾒｰﾀ･ﾃｰﾌﾞﾙ***************************
                .even
param_adr_table dc.l            param_table_1
param_table_1   dc.w            $0008
                dc.w            $8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ･ﾃｰﾌﾞﾙ**********************
                .even
exec_table      dc.l            oddeven

*************** FUNC MAIN *****************************
               .even
oddeven:        move.l          12(sp),a0
                move.l          #-1,d0
count           addq.l          #1,d0
                tst.b           (a0)+
                bne             count
                and.l           #1,d0
```

```
                move.l      d0,ret_param+6
                clr.l       d0
                lea.l       ret_param(pc),a0
                lea.l       error_comment(pc),a1
return          rts

                .even
ret_param       dc.w        0
                dc.l        0
                ds.l        1

error_comment   dc.b        "error",0

                end
```

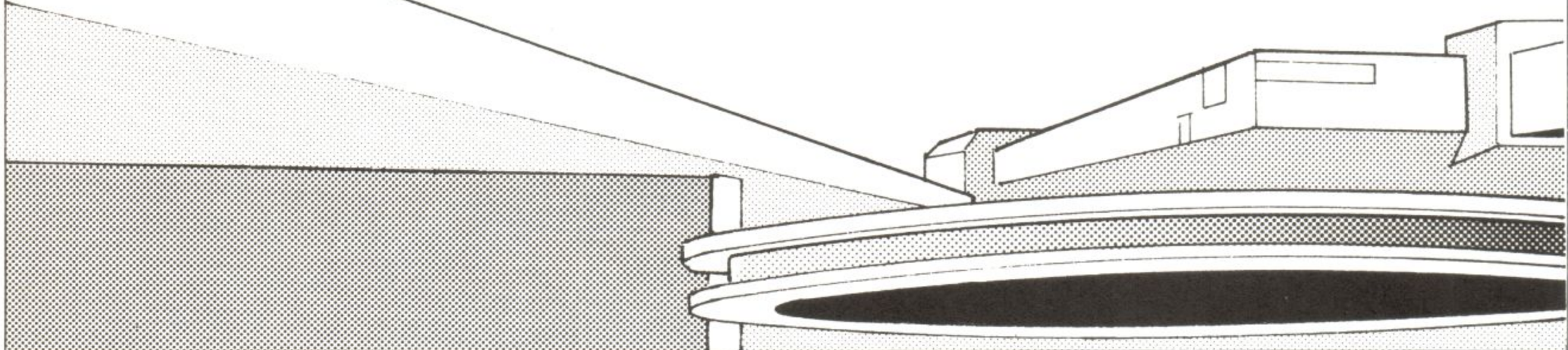
第4－6図　**完成した外部定義関数** oddeven

第4－6図が完成した外部定義関数のソースリストです。手順どおり実行されていれば，次にあげる**第4－7図**のリストを実行して次のような結果が出ると思います。ただし，出来上がった外部関数はBASIC.CNFに登録しておかなければなりません。

```
10 str a="X-BASIC",b="外部定義関数"
20 int x,y
30 x=oddeven(a):y=oddeven(b)
40 print x,y
50 end
run
1        0
Ok
■
```

第4－7図　oddeven**のサンプルプログラム**

このプログラムで，aは7文字でbは全角で6文字つまり半角文字として数えた場合12文字ということになります。従って，oddeven（a）は1，oddeven（b）は0ということになります。

4－3　外部定義関数の書式（その２）

4-3-1　配列を引数とする外部定義関数

4-2で紹介した外部定義関数では，引数としてstr型，戻り値としてint型で作りました。しかし，ユーザーの発想はとどまることを知りませんから，奇抜な考えや非常に優れたものなど色々あります。そのような中に，引数に配列を使いたい場合が出てきます。外部関数の中のAUDIO.FNCで出てきたa_rec，a_playなどがそうです。基本的には，前述の外部関数の定義法は付属の外部関数と同じ構造であるので，ユーザーが配列を引数とする関数を作成することも可能です。

それでは，実際に配列を引数とする外部定義関数を作ってみます。この関数の仕様書は**第4－8図**のようになります。関数のソースプログラムは**第4－9図**の通りです。

関数名：sum（　）
引数の数と型：int型の配列とint型
戻り値の型：int型
関数の内容：配列に与えられた数値の合計を計算して戻り値へ返します。関数のプログラムを簡素化するために，第２の引数として計算させる配列の数を与えます。

第4－8図　外部定義関数　sum（　）の仕様

```
**************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ************************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
```

```
adr_token        dc.l          token_table
adr_parameter    dc.l          param_adr_table
adr_exec         dc.l          exec_table
reserve          dc.l          0,0,0,0,0

*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ*****************************

token_table      dc.b          "sum",0,0

*************** ﾊﾟﾗﾒｰﾀ・ﾃｰﾌﾞﾙ***************************
                 .even
param_adr_table  dc.l          param_table_1

param_table_1    dc.w          $0032,$0002
                 dc.w          $8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ・ﾃｰﾌﾞﾙ***********************
                 .even
exec_table       dc.l          sum

*************** FUNC MAIN ******************************
                 .even
sum:             move.l        12(sp),a0    ‥‥ ポインターのポインター
                 move.l        22(sp),d2
                 lea.l         10(a0),a0    ‥‥ 配列の先頭アドレス（ﾎﾟｲﾝﾀｰ）
                 clr.l         d0
nextadd          move.l        (a0)+,d1
                 add.l         d1,d0
                 subq.l        #1,d2
                 cmp.l         #0,d2
                 bne           nextadd      ‥‥ d2が0になるまでループ
                 move.l        d0,ret_param+6
                 clr.l         d0
                 lea.l         ret_param(pc),a0
                 lea.l         error_comment(pc),a1
return           rts

                 .even
ret_param        dc.w          0
                 dc.l          0
                 ds.l          1

error_comment    dc.b          "error",0
```

```
                    end
```

第4－9図　外部定義関数sumのソースリスト

このプログラムを実行させるX-BASICのサンプルプログラムは，第4－10図の通りです。

```
10 int a,x=10
20 dim int d(9)={1,2,3,4,5,6,7,8,9,10}
30 a=sum(d,x)
40 print "total=",a
50 end
run
total= 55
Ok
█
```

第4－10図　sumの使用例

ご覧のように，引数として配列を渡すことにより，非常に簡略化された記述になります。この方法を使うことにより，たくさんの引数を必要とする関数には，配列を渡すことがいかに能率の良い方法かが分かります。

実際の配列の引き渡しについては，第5章に事例がありますので，それも参考にして下さい。

4-3-2 引数を戻り値として使う外部定義関数

定義関数で説明したグローバル変数とローカル変数で，引数を持たない関数の定義を解説しましたが，これとは反対に戻り値をダミーにしてしまい，答えを引数に返す方法があります。これは，外部関数のMOUSE.FNCのmsposでマウスの位置を引数に返す方法と同じです。これは，引数としてポインタ（引数として使った変数のメモリアドレス）を受け取り，結果をそのポインタへ返すものです。この方法を使うと，関数の基本概念であった引数に対する戻り値ということが成り立たなくなるほか，戻り値を複数にすることが出来る点に注目できます。

関数の中には，グラフィックス等によく使う座標計算では，平面ではx，yがあり，立体になればx，y，zと三つの座標系が必要になります。このような場合，ある条件のもとでその関数を通せば座標情報がx，yで返ってくるような関数も作ることが出来ます。こういう場合には，ダミーの戻り値を設定しなければならないのですが，それはそのまま使わなければいいのです。それでは，4-3-1で定義したsum（　）を改造して，引数に答えを返す方法を見てみます。

```
*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ***********************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ*****************************

token_table     dc.b            "sum1",0,0

*************** ﾊﾟﾗﾒｰﾀ・ﾃｰﾌﾞﾙ***************************
                .even
param_adr_table dc.l            param_table_1

param_table_1   dc.w            $0032,$0002,$0012
                dc.w            $8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ・ﾃｰﾌﾞﾙ***********************
                .even
exec_table      dc.l            sum1

*************** FUNC MAIN ******************************
               .even
sum1:           move.l        12(sp),a0
                move.l        22(sp),d2
                move.l        32(sp),a1･･･第３引き数のポインターを受け取る
                lea.l         10(a0),a0
                clr.l         d0
nextadd         move.l        (a0)+,d1
                add.l         d1,d0
                subq.l        #1,d2
                cmp.l         #0,d2
                bne           nextadd
                move.l        d0,(a1)････第３引き数のポインターへ答えを返す
```

```
                move.l      d0,ret_param+6
                clr.l       d0
                lea.l       ret_param(pc),a0
                lea.l       error_comment(pc),a1
return          rts
                .even
ret_param       dc.w        0
                dc.l        0
                ds.l        1

error_comment   dc.b        "error",0

                end
```

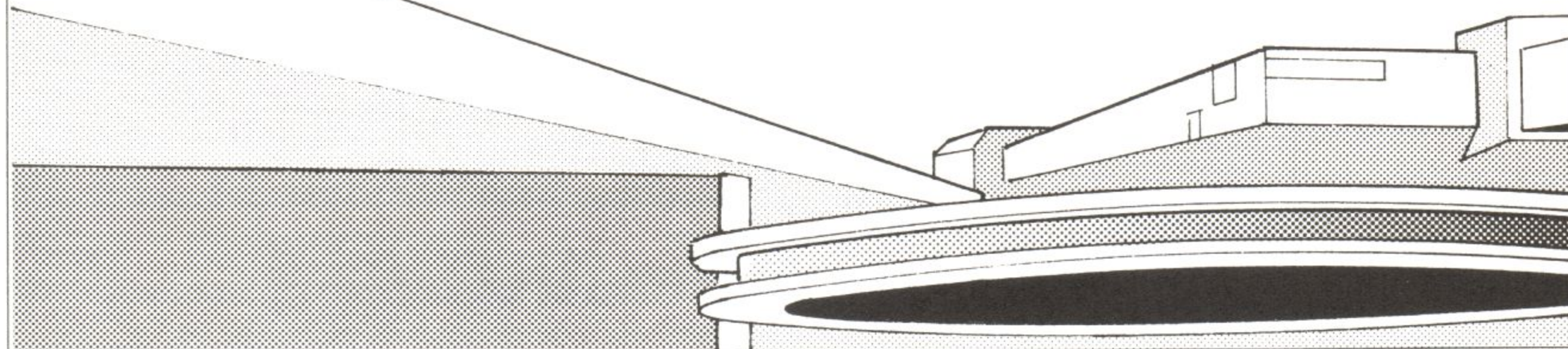
第4－11図　sum を sum 1 に改造する

第4-11図の様に，先程の関数に第3の引数（int 型でポインタ渡し）を付け加えます。そして，戻り値を返すのと同時に，3番目の引数として受け取った引数のポインタへも答えを返します。その結果を見るため，**第4-12図**の X-BASIC のプログラムを実行してみて下さい。ご覧の通り，次のような答えが確認出来るはずです。

```
10 int a,b,x=10
20 dim int d(9)={1,2,3,4,5,6,7,8,9,10}
30 a=sum1(d,x,b)  ························引き数が3つになっている
40 print "total1=",a
50 print "total2=",b
run
total1= 55
total2= 55
Ok
█
```

第4－12図　sum 1の実行例

ご覧のように，引数として渡したはずのｂに戻り値ａと同じ答えが格納されています。このように，ポインタとして引数を渡すことで，引数として渡す変数のメモリアドレスが分かります。前述の通り，配列の引数はポインタを渡していますから，複数の答えを返したい場合は答えを格納するための配列変数を渡してやればいいわけです。

4—4 システムコールとIOCSコール

X68000には非常に使い易い**IOCS**（Input Output Control System）があります。これは**IPL**（Initial Program Loader）と一緒に**ROM**（Read Only Memory）内にあり，電源を切っても内容は消えません。このIOCSには，Human68kやX-BASICでも使用している便利なマシン語サブルーチンがギッシリ詰っています。また，Human68kでサポートされているシステムコールもこのIOCSを使用してプログラムされています。

外部定義関数では，マシン語によりユーザーが独力で色々な処理を作らなければなりませんが，このIOCSやシステムコールを使用することで広範囲に渡り，X68000のギッシリ詰った機能を使いこなすことが出来るのです。

4-4-1 IOCSの使い方

IOCSコールを使う場合のカギは，trap　#15という命令です。これは，68000のマシン語命令の中でも特殊な命令で，X68000の場合，この命令を受け取るとデータレジスタ0（d0）に書き込まれている番号のサブルーチンへ処理を移し，その処理が終わるとtrap #15の次の命令からプログラムを続けます。

この番号（d0へ書き込まれる番号）のことをIOCSコール番号と言って200近くのサブルーチンがあります。このサブルーチンについては，第5章に外部定義関数で使用したものの詳細を説明していますが，この他のものについてはX68000活用研究I（塚越一雄著）に詳しく載っていますので参考にして下さい。

それでは，一般的なIOCSコールの方法を**第4—13図**に記述しますので，第5章の記述も含め参考にして下さい。

```
                .
                .        前のプログラムからの流れ
                .
moveq           #$xx,d0･･･ xx部分にＩＯＣＳコール番号が入る
（その他のレジスタをセット）
trap            #15
```

次の処理へ

第4－13図　IOCS使用法

このようにX-BASICは，ユーザーの能力に合わせてBASIC自身も進化していきます。また進化したBASICは記述も簡単になり，理想に近づいていきます。しかし，ユーザーが怠けているとBASICも進化せずそのままでいます。X-BASICとはそういうBASICなのです。

4-4-2 IOCSコールの実習

さて，ここで実際にIOCSコールを使用した外部定義関数を作ってみます。関数の内容は**第4－14図**の仕様書通りです。

関数名：cur_prn（ ）
引数の数と型：str型が一つとint型が二つ
戻り値の型：int型
関数の内容：X-BASICのステートメントのlocate,printと標準関数の文字列処理関数のstrlen（ ）を組み合わせたものです。引数に文字列と表示開始位置のx，yを渡します。戻り値には，引数として渡した文字列の数が返ります。ただし，この関数では表示出来るのは，文字型の変数か文字列だけです。

第4－14図 外部定義関数cur_prn（ ）の仕様

IOCSルーチンの中で，文字列を画面に表示するサブルーチンと，指定位置へカーソルを移動させるサブルーチンがあります。それぞれのIOCSルーチンのデータは**第4－15図**の通りです。

ＩＯＣＳ名：Ｂ＿ＰＲＩＮＴ
ＩＯＣＳコール番号：＄２１
指定レジスタ：ａ１.ｌ（ロングワード）に文字列先頭アドレスを格納する
処理内容：ａ１に格納されているアドレスから１文字ずつ現在のカーソル位置から表示していきヌルコード（０）を発見したところで終了する。

ＩＯＣＳ名：Ｂ＿ＬＯＣＡＴＥ
ＩＯＣＳコール番号：＄２３
指定レジスタ：ｄ１．ｗ・・・カーソル桁位置
　　　　　　　ｄ２．ｗ・・・カーソル行位置
処理内容：ｄ１に格納してある桁位置とｄ２に格納してある行位置により指定される場所へカーソルを移動する。

第4－15図　cur_prnに使用したIOCSのデータ

第4－16図がこの関数のソースリストです。仕様書の通り，関数名はcur_prnです。この_はアンダーバーと言ってスペースが使えないような場合に使いますが，第5章の事例で紹介する外部定義関数のソースリストの中で数多く出てきますので注意して下さい。

```
*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ**********************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ****************************

token_table     dc.b            "cur_prn",0,0

*************** ﾊﾟﾗﾒｰﾀ・ﾃｰﾌﾞﾙ**************************
                .even
param_adr_table dc.l            param_table_1
param_table_1   dc.w            $0008,$0002,$0012
                dc.w            $8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ・ﾃｰﾌﾞﾙ**********************
                .even
exec_table      dc.l            cur_prn

*************** FUNC MAIN *****************************
                even
cur_prn:        move.l        12(sp),a0･･･文字列のポインタアドレス
                move.l        22(sp),d1･･･カーソルx位置
                move.l        32(sp),d2･･･カーソルy位置
                move.l        #-1,d0
count           addq.l        #1,d0
                tst.b         (a0)+
                bne           count
                move.l        d0,ret_param+6
```

```
                    move.l          12(sp),a0

                    moveq           #$23,d0
                    trap            #15

                    moveq           #$21,d0
                    move.l          a0,a1
                    trap            #15

                    clr.l           d0
                    lea.l           ret_param(pc),a0
                    lea.l           error_comment(pc),a1
return              rts

                    .even
ret_param           dc.w            0
                    dc.l            0
                    ds.l            1

error_comment       dc.b            "error",0

                    end
```

第4－16図　cur_prn のソースリスト

この外部定義関数をX－BASICで実際に使用した例が，**第4－17図**です。ご覧のように，str 型に宣言された変数aには"X－BASIC"の7文字を代入してあります。カーソル位置の指定用に宣言されたint型の変数x，yには，それぞれwidth 64の画面でほぼ中央に位置する31と15を代入しています。戻り値として返ってくる文字数はzへ代入されます。

```
10 str a="X-BASIC"
20 int x=31,y=15,z
30 z=cur_prn(a,x,y)
40 locate 0,0,1
50 print "LENGTH=";z
60 end
```

第4－17図　cur_prn のサンプルプログラム

実行結果は画面中央に"X-BASIC"と表示され，画面左上にはLENGTH＝7と表示されていると思います。

さて，次に少し変わった実験をしてみます。今作った外部定義関数に，文字列変数ではない数

値型の変数を引数として実行させてみます。

```
10 /*str a="X-BASIC"
20 int x=31,y=15,z,a=300
30 z=cur_prn(a,x,y)
40 locate 0,0,1
50 print "LENGTH=";z
60 end
run
引数の型が違います……30行
Ok
█
```

第4－18図　まちがった使用法

この様に，引数は外部関数のヘッダ部分で決められたもの以外では，エラーになってしまいます。しかし，ステートメントのprint文は，文字列や数値などどちらも使用することが出来ます。こうして考えてみれば，print文が関数でなくステートメントであった訳が分かります。

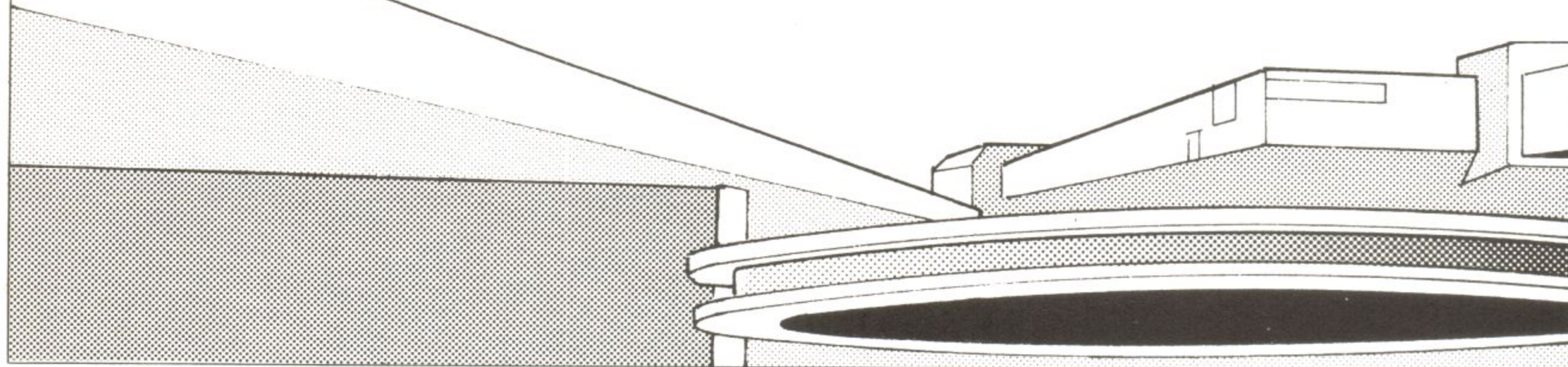
(注) 実際には関数に出来ないのではなくて，使い易さや，インタープリタ制作の時に作り易かったりなどにより選択されたのかも知れません。実際に，C言語ではBASICでのprint文に相当するprintfという関数が存在しています。

ワンポイントテクニック　**算術演算子**

X-BASICでは，皆さんが普段使用している算術演算(＋－×÷)の他に，色々な算術演算子があります。

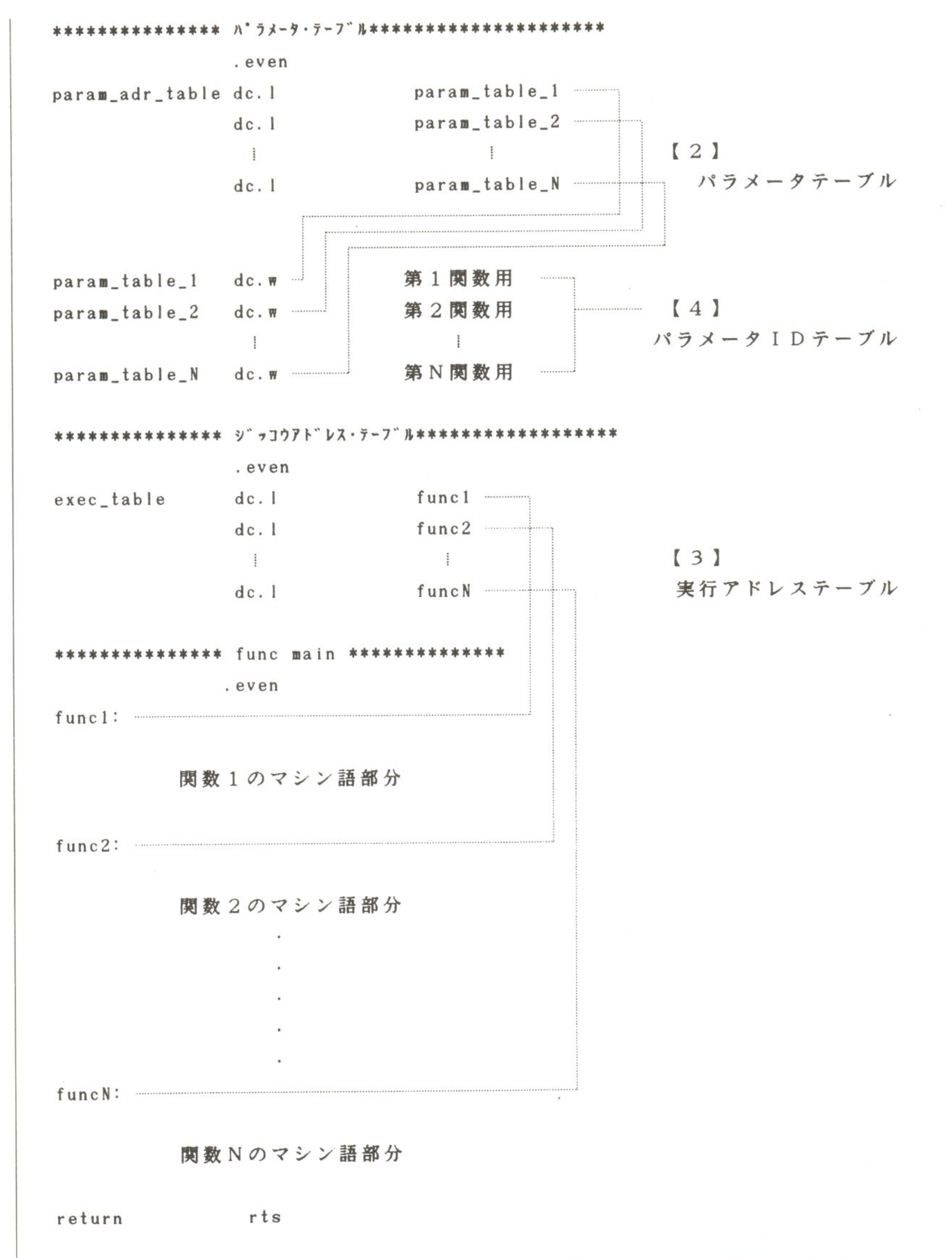
演　算　子	演　算　内　容	表　記　例
－	マイナス符号	－1，－X
＊／	掛け算，割り算	A＊B，C／D
＋－	足し算，引き算	A＋B，C－D
￥ mod	整数の除算，剰余	A￥B，C mod D
shr shl	右シフト，左シフト	A shr B，A shl B

演算の優先順位は，一般的な数学の法則に準じています。カッコを使用して，演算の順序を変えることも出来ます。

4－5　外部定義関数の書式（その3）

外部定義関数について色々と説明してきましたが，外部定義関数にとってヘッダの重要性が理解出来たことと思います。今までは，関数一つに対してこのヘッダが一つずつ付いていました。しかし，前述のように「外部定義関数はX-BASIC付属の外部関数と基本的に同じ構造をしている」としてありましたが，各外部関数は，関数群として＊＊＊．FNCのファイルの中に2個から多いもので20個近くもの関数を持っています（隠れ関数も入れればもっと多くなります）。

結論から言うと，ユーザーが定義する外部関数でも複数の関数を結合することが出来ます。1ファイルに関数を複合化するには，ヘッダに秘密があるのです。外部定義関数を1ファイルに複合化するヘッダは**第4－19図**の様になります。

```
*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ*****************

adr_init        dc.l        return
adr_run         dc.l        return
adr_end         dc.l        return
adr_system      dc.l        return
adr_break       dc.l        return
adr_input       dc.l        return
adr_reserve1    dc.l        return
adr_reserve2    dc.l        return

adr_token       dc.l        token_table ········ﾄｰｸﾝﾃｰﾌﾞﾙの位置
adr_parameter   dc.l        param_adr_table ····ﾊﾟﾗﾒｰﾀﾃｰﾌﾞﾙの位置
adr_exec        dc.l        exec_table ·········実行ｱﾄﾞﾚｽﾃｰﾌﾞﾙの位置
reserve         dc.l        0,0,0,0,0

*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ***********************

token_table     dc.b        "func1",0       ┐
                dc.b        "func2",0       ├─【1】
                 :            :             │   トークンテーブル
                dc.b        "funcN",0,0     ┘
```

```
**************** ﾊﾟﾗﾒｰﾀ･ﾃｰﾌﾞﾙ**********************
                 .even
param_adr_table  dc.l          param_table_1
                 dc.l          param_table_2
                  ⋮                  ⋮                    【2】
                 dc.l          param_table_N              パラメータテーブル

param_table_1    dc.w          第1関数用
param_table_2    dc.w          第2関数用                  【4】
                  ⋮               ⋮                       パラメータIDテーブル
param_table_N    dc.w          第N関数用

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ･ﾃｰﾌﾞﾙ******************
                 .even
exec_table       dc.l          func1
                 dc.l          func2
                  ⋮              ⋮                        【3】
                 dc.l          funcN                      実行アドレステーブル

*************** func main ***************
                .even
func1:

            関数1のマシン語部分

func2:

            関数2のマシン語部分
                  .
                  .
                  .
                  .
                  .
funcN:

            関数Nのマシン語部分

return           rts
```

第4－19図　複数の関数を一つのヘッダで定義する

インフォメーションテーブルにあげられている三つのテーブル(トークンテーブル，パラメータテーブル，実行アドレステーブル）はそれぞれ【1】，【2】，【3】の位置を示しています。

【1】トークンテーブル

トークンテーブルには，X-BASIC側でこの外部関数で使用される関数名を定義しています。関数名はご覧のように複数個を一度に設定することが出来ますが，関数名と関数名の間には，0（セパレータ）を置かなければなりません。また，トークンテーブル終了を示す0を最後の関数名の後にも置きますから，最後の関数名の後には"0"，"0"と二つのセパレータが入ります。この関数名の並びは，後に出てくるパラメータテーブルや，実行アドレステーブルの並びと同順にしておかなければなりません。

【2】パラメータテーブル

パラメータテーブルには，【4】のパラメータIDテーブルの位置を示すラベルを，トークンテーブルで並べた関数名の順で設定します。このパラメータIDテーブルには，関数の引数と戻り値の型や数を示すIDの並びを設定します。

【3】実行アドレステーブル

実行アドレステーブルには，トークンテーブルで関数名を置いた順で，マシン語プログラム本体の始まる位置をラベル名で置きます。

今のヘッダを使って今までに作ってきた外部定義関数を一つにまとめてみます。まず，今までに作ってきた外部関数のマシン語プログラム本体だけを**第4－20図～第4－22図**のように抜き出します。

```
                .even
oddeven:        move.l          12(sp),a0
                move.l          #-1,d0
count           addq.l          #1,d0
                tst.b           (a0)+
                bne             count
                and.l           #1,d0
                move.l          d0,ret_param+6
                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           error_comment(pc),a1
return          rts

                .even
ret_param       dc.w            0
                dc.l            0
                ds.l            1
```

```
error_comment   dc.b        "error",0

                end
```

第4－20図　oddeven のマシン語本体

```
                .even
sum:            move.l      12(sp),a0
                move.l      22(sp),d2
                lea.l       10(a0),a0
                clr.l       d0
nextadd         move.l      (a0)+,d1
                add.l       d1,d0
                subq.l      #1,d2
                cmp.l       #0,d2
                bne         nextadd
                move.l      d0,ret_param+6
                clr.l       d0
                lea.l       ret_param(pc),a0
                lea.l       error_comment(pc),a1
return          rts

                .even
ret_param       dc.w        0
                dc.l        0
                ds.l        1

error_comment   dc.b        "error",0

                end
```

第4－21図　sum のマシン語本体

```
                .even
cur_prn:         move.l        12(sp),a0
                 move.l        22(sp),d1
                 move.l        32(sp),d2
                 move.l        #-1,d0
count            addq.l        #1,d0
                 tst.b         (a0)+
                 bne           count
                 move.l        d0,ret_param+6
                 move.l        12(sp),a0

                 moveq         #$23,d0
                 trap          #15

                 moveq         #$21,d0
                 move.l        a0,a1
                 trap          #15

                 clr.l         d0
                 lea.l         ret_param(pc),a0
                 lea.l         error_comment(pc),a1
return           rts

                 .even
ret_param        dc.w          0
                 dc.l          0
                 ds.l          1

error_comment    dc.b          "error",0

                 end
```

第４－22図　cur_prn のマシン語本体

この三つの外部関数を合体させるわけですが，このソースリストをよく眺めてみると，同じ名前のラベルがあります。count，return，error_comment，ret_param です。このように今までは単独で作成してきているために同一関数内であれば重複しない限りどんなラベルでも使えました。しかし，一旦これらのプログラムを合体させるとなると重複して定義されているラベルは，アセンブル時にエラーとなってしまいます。複合化した外部定義関数を作る時には，ここが一番注意しなければならない所なのです。

さて，この重複したラベルの中にインフォメーションテーブルの中で設定されているラベルがあります。return というラベルがそうですが，これは，ヘッダ一つに対して一つしか設定することが出来ないので，マシン語プログラム本体の一番後ろに一つだけ置いて，どのマシン語プログラムも終了した後はここにブランチしてくるようにします。error_comment や ret_param もどのマシン語プログラムからでも共通に使えそうです。後は，重複しているラベルを別なラベル名にしてやればいいわけです。これらの諸注意に基づいて一つにまとめ，先程の要領でヘッダを付けたものが**第4-23図**です。

```
*******************************************************
*          external function program                  *
*       NAMED "GOTTANI"        by 宮原  哲也          *
*       build in oddeven & sum & cur_prn function     *
*******************************************************


*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ**********************


adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0
```

```
*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ******************************

token_table     dc.b            "oddeven",0
                dc.b            "sum",0
                dc.b            "cur_prn",0,0

*************** ﾊﾟﾗﾒｰﾀ・ﾃｰﾌﾞﾙ****************************
                .even
param_adr_table dc.l            param_table_1
                dc.l            param_table_2
                dc.l            param_table_3

param_table_1   dc.w            $0008,$8001
param_table_2   dc.w            $0032,$0002,$8001
param_table_3   dc.w            $0008,$0002,$0002,$8001

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ・ﾃｰﾌﾞﾙ***********************
                .even
exec_table      dc.l            oddeven
                dc.l            sum
                dc.l            cur_prn

*************** FUNC MAIN ******************************

               .even
oddeven:        move.l          12(sp),a0
                move.l          #-1,d0
count           addq.l          #1,d0
                tst.b           (a0)+
                bne             count
                and.l           #1,d0
                move.l          d0,ret_param+6
                clr.l           d0
```

```
                lea.l           ret_param(pc),a0
                lea.l           error_comment(pc),a1
                bra             return
*****************************************************
sum:            move.l          12(sp),a0
                move.l          22(sp),d2
                lea.l           10(a0),a0
                clr.l           d0
nextadd         move.l          (a0)+,d1
                add.l           d1,d0
                subq.l          #1,d2
                cmp.l           #0,d2
                bne             nextadd
                move.l          d0,ret_param+6
                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           error_comment(pc),a1
                bra             return
*****************************************************
cur_prn:        move.l          12(sp),a0
                move.l          22(sp),d1
                move.l          32(sp),d2
                move.l          #-1,d0
count1          addq.l          #1,d0
                tst.b           (a0)+
                bne             count1
                move.l          d0,ret_param+6
                move.l          12(sp),a0

                moveq           #$23,d0
                trap            #15
```

```
                  moveq          #$21,d0
                  move.l         a0,a1
                  trap           #15

                  clr.l          d0
                  lea.l          ret_param(pc),a0
                  lea.l          error_comment(pc),a1
return            rts
****************work area***************************
                  .even
ret_param         dc.w           0
                  dc.l           0
                  ds.l           1

error_comment     dc.b           "error",0

                  end
```

第4－23図　三つの関数をまとめた"GOTTANI"

この外部定義関数をBASIC.CNFに登録する時には，今までに登録してあったoddeven，sum，cru_prnを全て削除(BASIC.CNF上だけでよい)しておかなければなりません。これは，ヘッダ部分でこれまでに定義してきた関数名が，今，定義した関数名と重複するためです。

それでは，今作った外部定義関数を使って動作の確認をしてみます。この外部関数をテストするには，今までに紹介してきたサンプルプログラム(それぞれの外部定義関数をテストしたもの)でも確認することが出来ますが，それでは面白くないのでここで紹介するプログラム**第4－24図**でテストしてみます。

```
10 width 64
20 int total,i,x,y,z
30 dim int d(3)
40 str a,b,c
50 input "string a:",a
60 input "string b:",b
70 input "string c:",c
```

```
 80 cls
 90 if oddeven(a)=0 then x=0:y=0 else x=50:y=30
100 d(0)=cur_prn(a,x,y)
110 if oddeven(b)=0 then x=0:y=30 else x=50:y=0
120 d(1)=cur_prn(b,x,y)
130 if oddeven(c)=0 then x=28:y=0 else x=28:y=30
140 d(2)=cur_prn(c,x,y)
150 total=sum(d,3)
160 locate 20,15,1:print "total strings length=";total
170 for i=0 to 3000:next
180 end
```

第4－24図　GOTTANIを使ってみよう

このプログラムは，GOTTANI.FNCの中に入っている関数を全て使用したものです。なかなか"ごった煮"ということで，関連のない関数どうしなので，結び付けるのが大変ですが，入力された三つの文字列のそれぞれが奇数か偶数かで表示位置をかえています。その後，それぞれの文字数を加えたものを画面中央へ表示します。

これまでにいろいろと外部定義関数に付いて解説してきましたが，68000のマシン語を使用したものなので，分からない方もいると思います。しかし，ここで紹介した68000のマシン語は，数種類しか使用していません。一つずつ確認しながら覚えていくことでマシン語プログラムに開眼することもあります。他人のプログラムを読む（解析する）ことから第一歩が始まります。

ワンポイントテクニック　X68000隠し機能・入門編

X68ユーザーなら知っている方も多いと思われますが，有名な隠し機能です。まだ知らない人は是非覚えておきましょう。

- OPT.1とOPT.2を同時に押すと10進・16進電卓が使える
- OPT.2を押しながら電源ONでRS232Cからデバッガが使える
- BASICでは，LISTはL．RUNはR．などと省略形が使える
- 記号入力から右3個のキーを押しながら背面電源ONでテストモード
- （OSの）SWITCH　?↵で拡張スイッチ使用方法が表示される
- RAMディスク定義で#GM128ならGRAMとメモリ両方使える
- BASICでSTR A$のようにすれば$付きの名前も利用できる
- DIRはもちろんREN,DEL,TYPEでもワイルドカードOK

ワンポイントテクニック　マニュアルに載っていないコマンド＆標準関数

マニュアルに載っていないコマンド

【KILL】

書　式	kill ”ファイル名”
機　能	ファイルを削除します。

【NAME】

書　式	name ”新ファイル名”，”旧ファイル名”
機　能	ファイル名の変更を行います。

【CHDIR】

書　式	chdir ”ディレクトリ名”
機　能	ディレクトリを変更します。

【CHDRV】

書　式	chdrv ”ドライブ名”
機　能	ドライブを変更します。

【SEARCH】

書　式	search ”＊＊＊＊＊”（＊＊＊＊＊は，探したい文字列）
機　能	探したい文字を含んだプログラム行を表示します。

【LSEARCH】

書　式	lsearch ”＊＊＊＊＊”（＊＊＊＊＊は，探したい文字列）
機　能	探したい文字を含んだプログラム行をプリンタに出力します。

マニュアルに載っていない標準関数

【FDELETE】

書　式	f ＝fdelete（fn）
引　数	str fn　　fn……ファイル名
機　能	fnで指定されたファイルを削除します。
戻り値	int f　　f ＝ 0の時　削除できた場合 f ＝－1の時　エラーが発生した場合

【FRENAME】

書　式	f ＝frename（old，new）
引　数	str old，new old ………変更したいファイル名 new………新しく付けるファイル名
機　能	ファイル名を変更します。
戻り値	int f　　f ＝ 0の時　削除できた場合 f ＝－1の時　エラーが発生した場合

第5章

X-BASICで役立つ外部定義関数・事例集

5-0　事例集の書式

5-1　テレビコントロール関数「TVCTRL」

5-2　スプライト・グラフィック・テキストのプライオリティ設定関数「PRI」

5-3　マウスカーソルの定義関数「MSCSET」

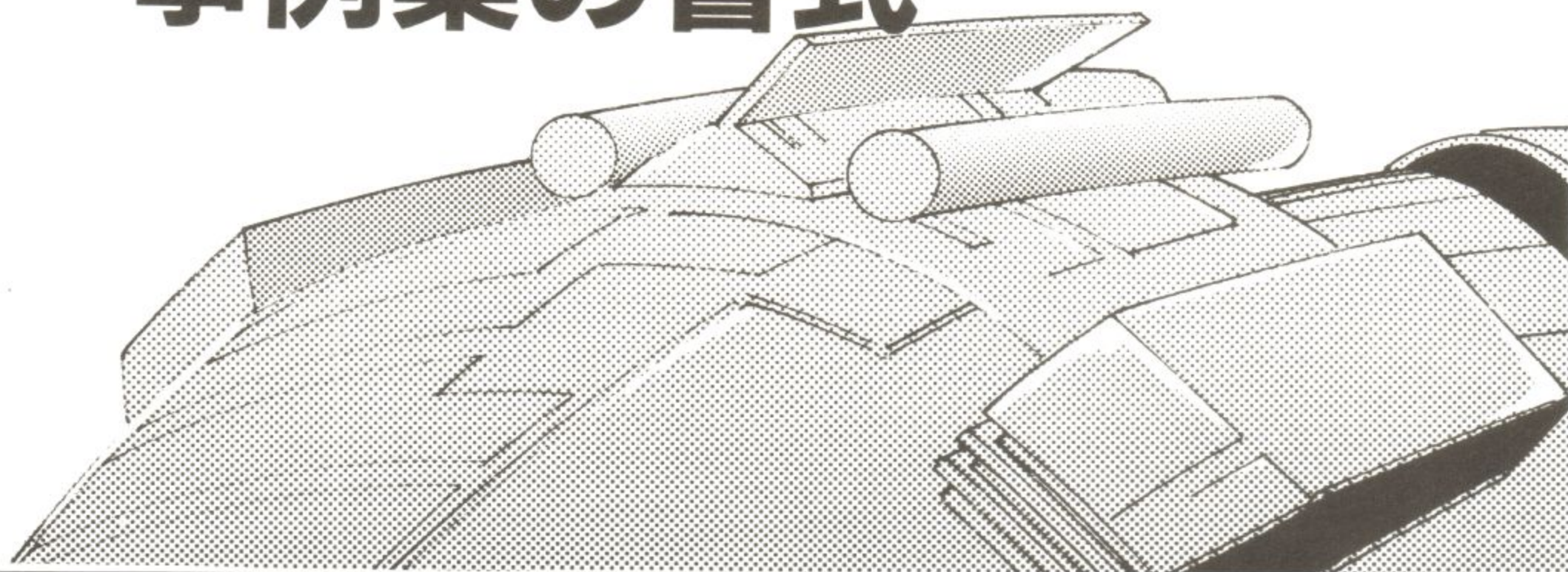
5-4　使用したいマウスカーソル番号を設定する関数「MSCURSOR」

5-5　複合関数「XLINE」「TLINE」「TCLS」

5-6　XORタイプのCIRCLE「XCIRCLE」

5-7　検索機能付FILES関数「FFILES」

5-8　ポップアップメニュー関数「POP_UP」

5-0　事例集の書式

第4章で外部定義関数を実習しましたが，ここでは実際にユーザーがX-BASICで使用して便利な外部定義関数を紹介します。X68000のハードウエアの深いところへ入り込むものもありますから，ソースプログラムの打ち込みには十分注意してください。また，出来上がった外部定義関数は，basic.cnfに登録しなければなりません。

それでは，ここでは外部定義関数を紹介するにあたっての書式について説明します。

書式1……**関数名**（ファイル名と兼用）

章の副題の後に付いているものが関数名です。これは，アルファベットの大文字で書いてありますが，定義する時は大文字，小文字どちらでも使用出来ます。また，ソースプログラムを打ち込む時に，ED.Xを起動すると思いますが，ED.Xの起動時にもこの関数名を使用して下さい。必ずしも，この関数名でファイル登録しなくてもよいのですが，basic.cnfをのぞいた時どんな関数が登録されているかが一目で分かりますので，これからも，自分で作った関数もファイル名と同じにしておくことをお薦めします。

書式2……**外部定義関数の仕様書**（第4章参照）

第4章で紹介した外部定義関数作成時に役立つ仕様書を少し改造したもので，関数の内容紹介をします。内容紹介も第3章の記述の方法に統一されています。

書式3……**ソースプログラム**（ED.Xでそのまま入力してください。）

ソースプログラムは，ED.Xでそのまま入力していけば打ち間違えさえなければ，動作すると思います。打ち込んだ後は，もちろんAS.Xによりアセンブルし，LK.Xでリンクした後は，関数名の＊＊＊.Xを＊＊＊.FNCにして下さい。

書式4……**使用例と解説**（X-BASICでの使用例）

さて，せっかく出来上がった外部定義関数も，使われて初めてその役目を果たします。ここでは，簡単なX-BASICによる使用例を紹介します。そのプログラムを実行した結果が画面上で表示されるものについては極力X-BASICリスト後にrun（実行）してその結果を載せてあります。視覚的に見せることが出来なかったものについても結果を克明に書きますので，デバッグの資料にして下さい。

この章で紹介する外部定義関数は，**第5－1図**にあげるものです。出来るだけ簡単なものから順に載せています。理解の度合いに併せて，関数作成に役立てて下さい。

〔関数名〕		〔機　　能〕	〔難易度〕
●TVCTRL		テレビ制御関数	1
●PRI		プライオリティ設定関数	2
●MSCSET		マウスカーソル定義関数	3
●MSCURSOR		使用マウスの選択関数	1
●XLINE	複合関数	XORライン(グラフィック)関数	4
●TLINE	複合関数	ORライン(テキスト画面)関数	5
●TCLS	複合関数	テキストページクリア関数	2
●XCIRCLE		XORサークル(グラフィック)	5
●FFILES		検索機能付きFILES関数	5
●POP_UP		ポップアップメニュー関数	5

第5－1図　紹介する外部定義関数の難易度

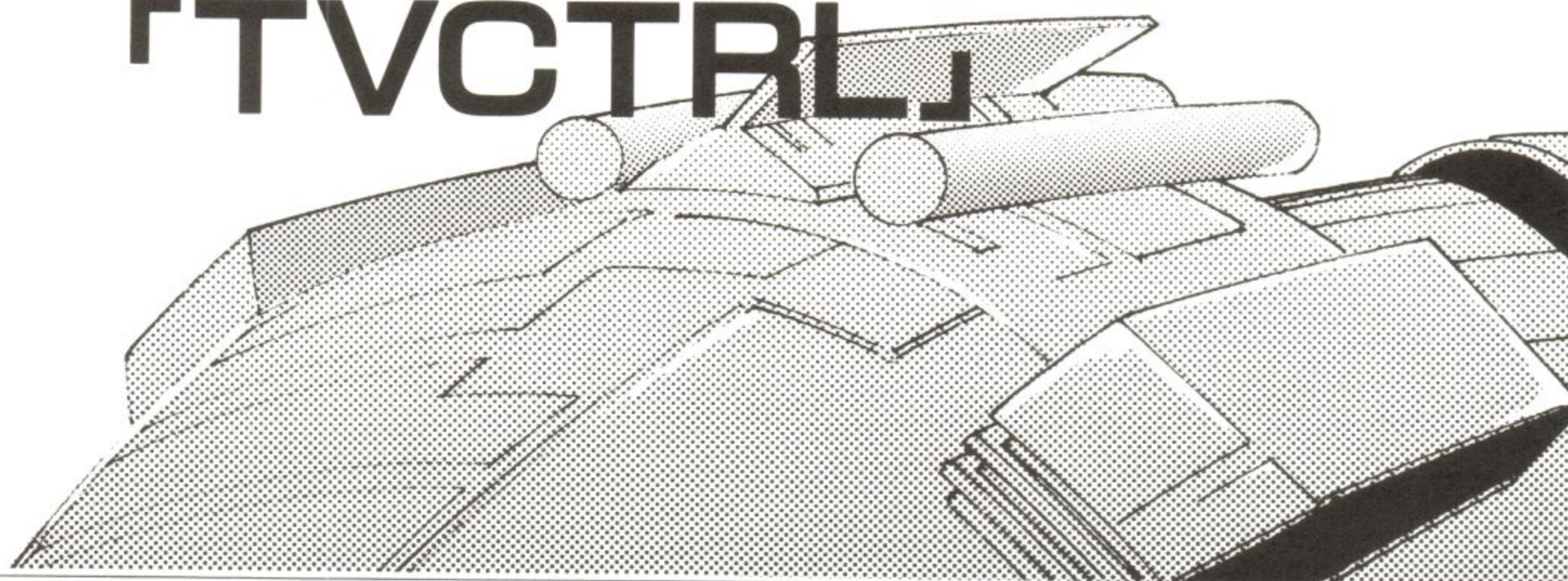
ワンポイントテクニック　関係演算子

関係演算子は，ｉｆ文での判定やwhile，until等での条件式でよく使われます。X-BASICでは，次のような関係演算子の使用が許されています。

関係演算子	内　容	使用例
＝	等しい	A＝B
＜＞	等しくない	A＜＞B
＜	小さい	A＜B
＞	大きい	A＞B
＜＝	～以下	A＜＝B
＞＝	～以上	A＞＝B

演算結果が正しければ－1，正しくないと0となります。これらの結果は，関係演算子を含めた式全体の値としても使用出来ます。

　例　　A＝（X＞Y）カッコ内が正しければA＝－1，正しくなければA＝0となります。

5-1 テレビコントロール関数「TVCTRL」

仕様書１

関数名：tvctrl (n)
引数の数と型：int n（指定はIOCSコールのレジスタと同じ）
戻り値：ナシ
内容：引数で与えた制御コードにより，テレビ（CZ600Dだけに使用可能）をコントロールします。

リスト１

```
*************************************************************
*                                                           *
*        BASIC EXTERNAL FUNCTION PROGRAM LIBRARY            *
*                                                           *
*                 月刊マイコン　別冊                         *
*                                                           *
*              昭和６２年９月３０日　制作                    *
*                                                           *
*           PROGRAMED & Typed by   深沢　幸三               *
*                                                           *
*       〈〈 TVCTRL 〉〉  (Ver1.0)                           *
*                                                           *
*************************************************************


_tvctrl        equ      $000C
```

```
int_val         equ     $0002
int_ret         equ     $8001

adr_init        dc.l    return
adr_run         dc.l    return
adr_end         dc.l    return
adr_system      dc.l    return
adr_break       dc.l    return
adr_input       dc.l    return
adr_reserve1    dc.l    return
adr_reserve2    dc.l    return
adr_token       dc.l    token_table
adr_parameter   dc.l    param_adr_table
adr_exec        dc.l    exec_table
reserve         dc.l    0,0,0,0,0


token_table     dc.b    "tvctrl",0,0,0

                .even

param_adr_table dc.l    param_table

param_table     dc.w    int_val,int_ret


                .even

exec_table      dc.l    tvctrl

                .even

tvctrl:         move.l  12(sp),d1
                moveq   #_tvctrl,d0
```

```
                trap    #15

                clr.l   d0
                lea.l   ret_param(pc),a0
                lea.l   msg_error(pc),a1

return          rts

                .even

ret_param       dc.w    0
                dc.l    0
                dc.l    1

msg_error       dc.b    "Error",0
```

この関数に使用したIOCSコールは，次のようなものです。このIOCSコールは，レジスタd1にロングワードのデータを下記のような指定でセットし，IOCSコール番号をd0にセットしてtrap ＃15を実行すると指定した作業を実行するものです。

IOCSコール番号：$OC

IOCS名：TVCTRL

指定レジスタ：d1.l	$01/$02/$03	ボリュームを上げる/下げる/標準にする
	$04	チャンネルコール
	$05	テレビ画面
	$06	音声ミュート
	$07/$0D	電源ON/OFF
	$08	テレビ/コンピュータ
	$09	テレビ/外部，コンピュータノーマル/オーバー
	$0A	コントラストノーマル
	$0B/$0C	チャンネルアップ/ダウン
	$0E	電源ON/OFF切り替え
	$0F	スーパー1
	$10～$1B	チャンネル1～12
	$1C	テレビ画面（$05）
	$1D	コンピュータ画面（$05＋$08）
	$1E	スーパー1（$05＋&0F）

$1F　　　　スーパー２（$05＋&0F＋&0A）
$20＋上記　　電源ONと上記ファンクションの実行

第５－２図に示すプログラムは，このtvctrl（　）を実際に使用してみたプログラムです。このプログラムでは，プログラムが進むにつれていろいろなテレビコントロールをしていくものです。この関数を使えば，目覚し時計がわりに起きたい時間にディスプレイに電源が入り，音量を少し大き目にして起こしてくれたり，見たい番組が始まる時間にチャンネルを替えてくれたりなどいろいろと応用できそうです。

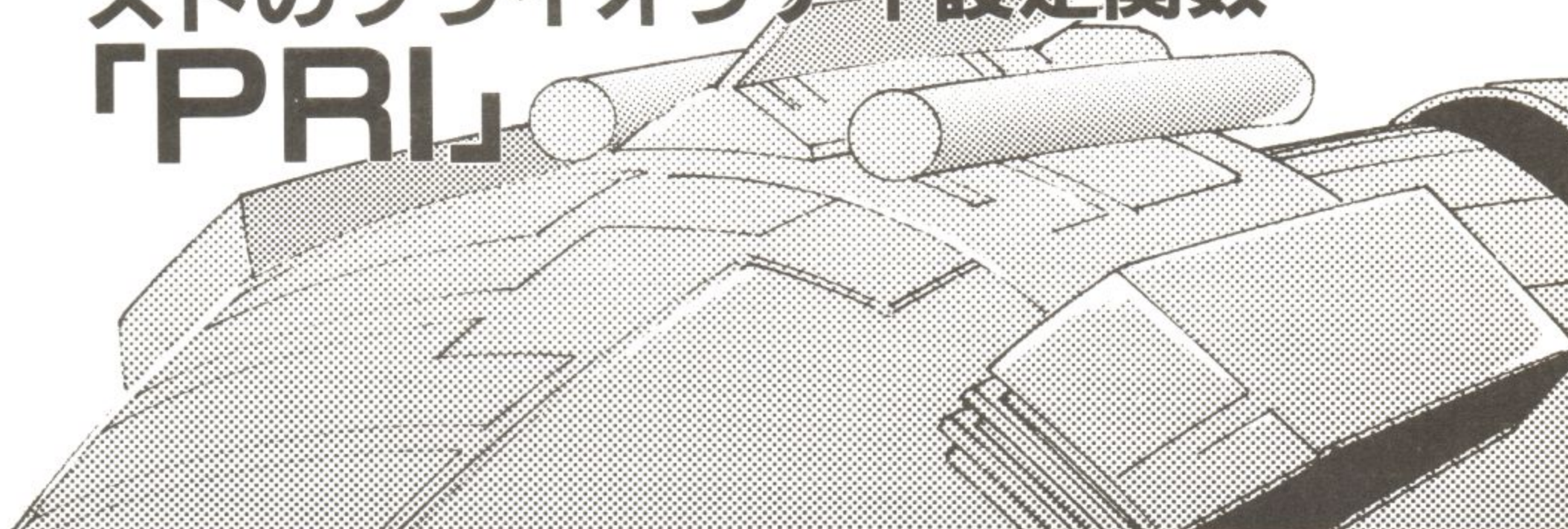
```
 10 /* tvctrl test
 20 int i
 30 tvctrl(5) :     /* Change TV
 40 tvctrl(6) :     /* Sound mute
 50 for i=1 to 15
 60    tvctrl(1) : /* Volume up
 70 next
 80 for i=0 to 20000
 90 next
100 for i=1 to 30
110    tvctrl(2) : /* Volume down
120 next
130 for i=0 to 20000
140 next
150 tvctrl(3) :     /* Volume normal
160 tvctrl(&HE) :   /* TV off
170 for i=0 to 50000
180 next
190 print"Ｔｈａｔ’ｓ　ｙｅｔ！"
200 tvctrl(&HE) :   /* TV on
210 for i=0 to 50000
220 next
230 tvctrl(8) :     /* Change X68000
240 end
```

第５－２図　tvctrlを使用したサンプルプログラム

第5－2図は，tvctrl関数を使用したディスプレイコントロールプログラムです。流れは，次のようになっています。

1．コンピュータ画面からテレビ画面に切り替える。
2．音声ミュート（消音）
3．徐々にボリュームを上げる。
4．徐々にボリュームを下げる。
5．ボリュームを標準にする。
6．ディスプレイの電源を切る。
7．コンピュータ画面にメッセージを出力する。
8．ディスプレイの電源を入れる。
9．テレビ画面からコンピュータ画面に切り替える。

以上の処理の間にその状態をしばらく保つためにfor～nextの空ループを入れて時間を稼いでいます。

ワンポイントテクニック　REM 文を活用しよう！

プログラムを作っていくうちにだんだん大きなプログラムになり，まとまりがつかなくなってしまうことがあります。従来の BASIC に比べ X-BASIC は構造化されたプログラムを記述することが出来るようになりましたが，なかなか慣れるまでは苦労も多いと思います。そんな時にプログラム中へREM 文を書き込んでおくと非常にわかり易くなります。

REM 文は／＊と記述しますが，この後に書く文字等はプログラムとして解釈されないためにどんな事を書いてもかまいません。自由に書き込んで下さい。ただし，あまりプログラムの中に REM 文が多すぎると逆にプログラム自身が読みにくくなりますので注意が必要です。できれば，定義関数の先頭や難しい処理をしよう思っている部分等に限定した方がいいでしょう。

また，SAVE したプログラムはファイルネームでしかどんなプログラムなのかが分かりません。LOAD した後 LIST した時，プログラムの先頭にどんなプログラムで，いつ，誰が作ったかなどを記入しておくとすぐにそのプログラムについてわかるので，プログラムの先頭に REM 文を書き込むことを習慣にした方がいいと思います。

5-2 スプライト・グラフィック・テキストのプライオリティ設定関数「PRI」

仕様書2

関数名：PRI（n）
引数の数と型：int n
戻り値：ナシ
内容：引数で指定したプライオリティにセットする。
　　引数の指定　1：スプライト＜テキスト＜グラフィック
　　　　　　　　2：テキスト＜スプライト＜グラフィック
　　　　　　　　3：スプライト＜グラフィック＜テキスト
　　　　　　　　4：テキスト＜グラフィック＜スプライト
　　　　　　　　5：グラフィック＜テキスト＜スプライト
　　　　　　　　6：グラフィック＜スプライト＜テキスト

リスト2

```
*****************************************************
*                                                   *
*      BASIC EXTERNAL FUNCTION PROGRAM LIBRARY      *
*                                                   *
*              月刊マイコン　別冊                   *
*                                                   *
*          昭和６２年１１月３日　制作               *
*                                                   *
*       PROGRAMED & Typed by　　宮原　哲也          *
*                                                   *
*        〈〈 ＰＲＩ 〉〉　（Ｖｅｒ１．０）         *
*                                                   *
*****************************************************
```

```
_pri            equ             $e82500
_super          equ             $ff20
                .text
********************************************

int_val         equu            $0002
int_ret         equ             $8001

*********************************************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

**********************************************

token_table     dc.b            "pri",0,0

**********************************************
                .even

param_adr_table dc.l            param_table

param_table     dc.w            int_val
                dc.w            int_ret
```

```
***********************************************
                .even
exec_table      dc.l            pri

***********************************************
                .even
pri:
                move.l          12(sp),hikisuu

                clr.l           -(sp)
                dc.w            _super
                addq.l          #4,sp
                move.l          d0,sspbuf
                move.l          usp,a0
                move.l          a0,uspbuf

                bsr             priority

                move.l          uspbuf,a0
                move.l          a0,usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp

                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

return          rts

                .even
ret_param       dc.w            0
                dc.l            0
                dc.l            1
```

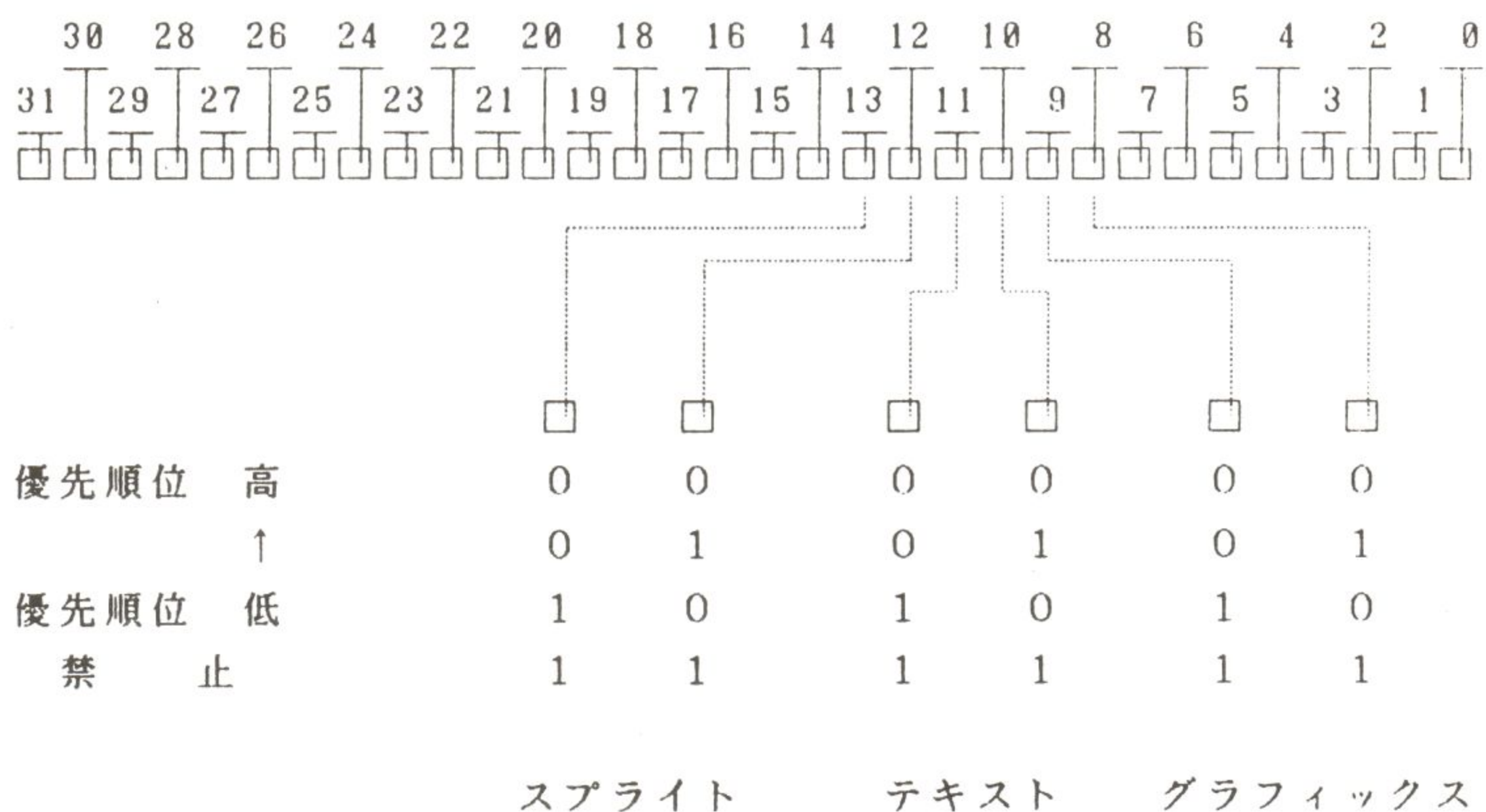
```
msg_error       dc.b        "外部関数エラーです",0

sspbuf          dc.l        0

uspbuf          dc.l        0

hikisuu         dc.l        0

*****************************************************
                .even
priority:
                move.l      hikisuu,d0
                cmp.l       #$1,d0
                beq         pri1
                cmp.l       #$2,d0
                beq         pri2
                cmp.l       #$3,d0
                beq         pri3
                cmp.l       #$4,d0
                beq         pri4
                cmp.l       #$5,d0
                beq         pri5
                cmp.l       #$6,d0
                bra         pri6
                move.l      #-1,d0
                move.l      d0,ret_param+6
                rts

pri1            move.w      #$2400,d1
                bra         cont
pri2            move.w      #$1800,d1
                bra         cont
pri3            move.w      #$2100,d1
                bra         cont
```

```
pri4        move.w      #$0900,d1
            bra         cont
pri5        move.w      #$0600,d1
            bra         cont
pri6        move.w      #$1200,d1
cont        move.w      _pri,d0
            and.w       #$c0ff,d0
            or.w        d0,d1
            move.w      d1,_pri
            clr.l       d0
            move.l      d0,ret_param+6
            rts

            .end
```

このプログラムでは，IOCSコールを使用していません。直接画面コントロール用LSIの内容を書き換えています。X68000のバス（アドレスやデータ）には数多くのLSIがつながっています。その中に画面を制御するものがありますが，そのLSIの機能の一部に，グラフィック，テキスト，スプライトのプライオリティを決定する部分があります。そこを直接書き換えて，それぞれの画面の優先順位を決めています。その一部分というのは次の通りです。

X68000のアドレスバスのE82500H番地の内容

```
 30  28  26  24  22  20  18  16  14  12  10   8   6   4   2   0
31  29  27  25  23  21  19  17  15  13  11   9   7   5   3   1
```

	スプライト		テキスト		グラフィックス	
	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
優先順位　高	0	0	0	0	0	0
↑	0	1	0	1	0	1
優先順位　低	1	0	1	0	1	0
禁　止	1	1	1	1	1	1

それぞれの2ビットずつで優先順位を設定しますが，同じ優先順位を設定することは出来ません。

```
 10 /* pri test
 20 int x,i,j
 30 int r=200,p=5
 40 float y
 50 str k
 60 dim str md(6)={ "","GR>TX>SP","GR>SP>TX",
 70                   "TX>GR>SP","SP>GR>TX",
 80                   "SP>TX>GR","TX>SP>GR"}
 90 dim char spr(255)={
100     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
110     0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,0,
120     0,10,9,9,9,9,9,9,9,9,9,9,9,9,10,0,
130     0,10,9,8,8,8,8,8,8,8,8,8,8,9,10,0,
140     0,10,9,8,7,7,7,7,7,7,7,7,8,9,10,0,
150     0,10,9,8,7,6,6,6,6,6,6,7,8,9,10,0,
160     0,10,9,8,7,6,5,5,5,5,6,7,8,9,10,0,
170     0,10,9,8,7,6,5,4,4,5,6,7,8,9,10,0,
180     0,10,9,8,7,6,5,4,4,5,6,7,8,9,10,0,
190     0,10,9,8,7,6,5,5,5,5,6,7,8,9,10,0,
200     0,10,9,8,7,6,6,6,6,6,6,7,8,9,10,0,
210     0,10,9,8,7,7,7,7,7,7,7,7,8,9,10,0,
220     0,10,9,8,8,8,8,8,8,8,8,8,8,9,10,0,
230     0,10,9,9,9,9,9,9,9,9,9,9,9,9,10,0,
240     0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,0,
250     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
260 screen 1,3,1,1
270 sp_init()
280 sp_disp(1)
290 sp_def(1,spr,1)
300 for i=1 to 15
310     sp_color(i,hsv(10,31,31-i*2))
320 next
```

```
330 for i=1 to 15
340     fill(i*15,0,i*15+15,511,hsv(96,31,i*2))
350 next
360 for i=0 to 15
370     for j=0 to 15
380        locate j*4,i*2:print"★   ";
390     next
400 next
410 locate 45,0:print md(5);" pri( 5 )"
420 while 1
430        for x=-200 to 200
440        k=inkey$(0)
450            if k<>"" then pri_set()
460            y=sqr(r*r-x*x)
470            sp_move(1,x+256,y+256,1)
480        next
490        for x=-200 to 200
500        k=inkey$(0)
510            if k<>"" then pri_set()
520            y=sqr(r*r-x*x)
530            sp_move(1,256-x,256-y,1)
540        next
550 endwhile
560 end
570 func pri_set()
580      p=p+1
590      if p=7 then p=1
600      locate 45,0:print md(p);" pri(";p;")"
610      pri(p)
620 endfunc
```

第５－３図　priを使用したサンプルプログラム

このプログラムは，実行後しばらくするとディスプレイにグラフィック画面によるグラディエーションが左半分に表示され，テキスト画面に全角文字の記号で星が表示され，スプライト画面には赤いブロックが移動（楕円運動）を始めます。この状態で画面右上には，現在のスプライト，テキスト，グラフィックの優先順位が表示されています。ここでキー（何でもよい）を押すとこの優先順位が一つずつ変わっていきます。各画面が手前，中間，後ろに変化することを確認して下さい。

ワンポイントテクニック　スーパーインポーズについて

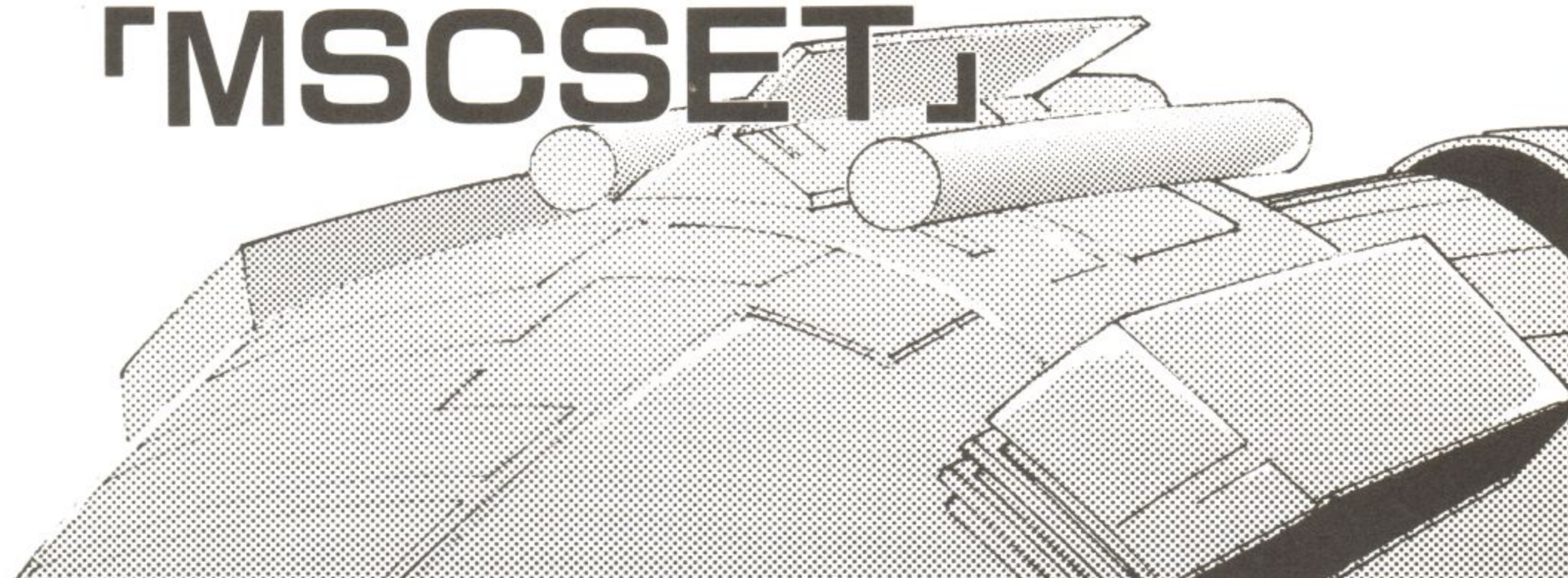
X68000では，X1から好評だったスーパーインポーズ機能をそのまま引き継いでいます。スーパーインポーズ機能というのは，テレビ画面にコンピュータ画面を重ね合わせて表示出来るということです。実際にこの機能を使用する場合には，若干の手続きを必要とします。これは，テレビ画面を構成する同期信号とコンピュータ画面を表示するための同期信号が違うためで，これによりスーパーインポーズ出来るコンピュータ画面も制約を受けます。

スーパーインポーズすることの出来るのは，低解像度の画面だけなので，SCREEN 命令で，３番目の引数は０としなければなりません。また，表示サイズも512×512以下でなければなりません。

では，実際にスーパーインポーズ画面にしてみます。次の手順で実施して下さい。

手順　（１）SCREEN　1，2，0，1とダイレクトモードでタイプする。
　　　（２）OPT.2キーを押しながら，テンキーの[+]を押す。

以上で画面にはテレビの画面とコンピュータ画面の合成が映し出されているはずです。テレビを見ながらプログラムする人には重宝なものですが，コンピュータ画面はテレビ画面よりはみ出して表示されているため残念ながら行番号や左端にあるカーソルが見えません。どうやら，プログラムを作ってから楽しむほかはないようです。例えば，ビデオカメラで撮った絵に文字を挿入するとかタイトル画面を作るとか……使い方はあなた次第です。

5-3 マウスカーソルの定義関数「MSCSET」

仕様書3

関数名：mscset（pn, px, py, na）

引数の数と型：int pn, px, py

pn……マウスカーソルNo.

px……パターンの左隅からマウス座標までの距離

パターンの左隅をマウス座標にする場合は0

パターンの右隅をマウス座標にする場合は15

py……パターンの上隅からマウス座標までの距離

パターンの上隅をマウス座標にする場合は0

パターンの下隅をマウス座標にする場合は15

na……マウスカーソルのパターンデータを格納しているint型の配列の名前

（要素数32個）

パターンの影にするデータ…（0～15）

パターン表示データ…………（16～31）

戻り値：ナシ

内容：int型の配列で指定した形のマウスカーソルを指定したカーソル番号に定義します。

注）設定出来るカーソルは一つだけです。

リスト3

```
***********************************************************
*                                                         *
*        BASIC EXTERNAL FUNCTION PROGRAM LIBRALY          *
*                                                         *
*                月刊マイコン　別冊                       *
*                                                         *
*             昭和６２年６月２２日　制作                  *
*                                                         *
*         PROGRAMED & Typed by　　深沢　幸三              *
*                                                         *
*     〈〈 MOUSE  CURSOR  SET 〉〉                        *
*                                                         *
***********************************************************

MS_PATST        equ     $007a

int_val         equ     $0002
int_vp          equ     $0032
int_ret         equ     $8002

adr_init        dc.l    return
adr_run         dc.l    return
adr_end         dc.l    return
adr_system      dc.l    return
adr_break       dc.l    return
adr_input       dc.l    return
adr_reserve1    dc.l    return
adr_reserve2    dc.l    return
adr_token       dc.l    token_table
adr_parameter   dc.l    param_adr_table
adr_exec        dc.l    exec_table
reserve         dc.l    0,0,0,0,0
```

```
token_table      dc.b     "mscset",0,0,0

                 .even

param_adr_table  dc.l     param_table

param_table      dc.w     int_val,int_val,int_val
                 dc.w     int_vp,int_ret

                 .even

exec_table       dc.l     mscset

                 .even

mscset:          move.l   12(sp),d0
                 move.l   22(sp),d1
                 move.l   32(sp),d2
                 move.l   42(sp),a0

                 lea.l    10(a0),a0

                 move.w   d0,pn
                 move.w   d1,buffer_adr1
                 move.w   d2,buffer_adr2
                 move.l   #32,d0
                 lea.l    pattern,a1
loop             move.l   (a0)+,d5
                 move.w   d5,(a1)+
                 subq.l   #1,d0
                 cmp.l    #0,d0
                 bne      loop

                 lea.l    buffer_adr1,a1
```

```
                move.w  pn,d1
                moveq   #MS_PATST,d0
                trap    #15

                clr.l   d0
                lea.l   ret_param(pc),a0
                lea.l   msg_error(pc),a1

return          rts

pn              dc.w    0
buffer_adr1     dc.w    0
buffer_adr2     dc.w    0
pattern         ds.w    16
                ds.w    16

                .even

ret_param       dc.w    0
                dc.l    0
                dc.l    1

msg_error       dc.b    "Error",0
```

このプログラムで使用したIOCSコールは，次の通りです。

IOCSコール番号：＄7a

IOCS名：MS_PATST

指定レジスタ：d1.w＝カーソルの番号

a1.1＝パターンのデータ格納アドレス

00（a１）.w＝パターンの左隅からマウス座標までの距離

パターンの左隅をマウス座標にする場合は０

パターンの右隅をマウス座標にする場合は15

02（a１）.w＝パターンの上隅からマウス座標までの距離

パターンの上隅をマウス座標にする場合は０

パターンの下隅をマウス座標にする場合は15

04（a 1）.w＝パターンの影にするデータ（16ワード以上）

20（a 1）.w＝パターン表示データ（16ワード以上）

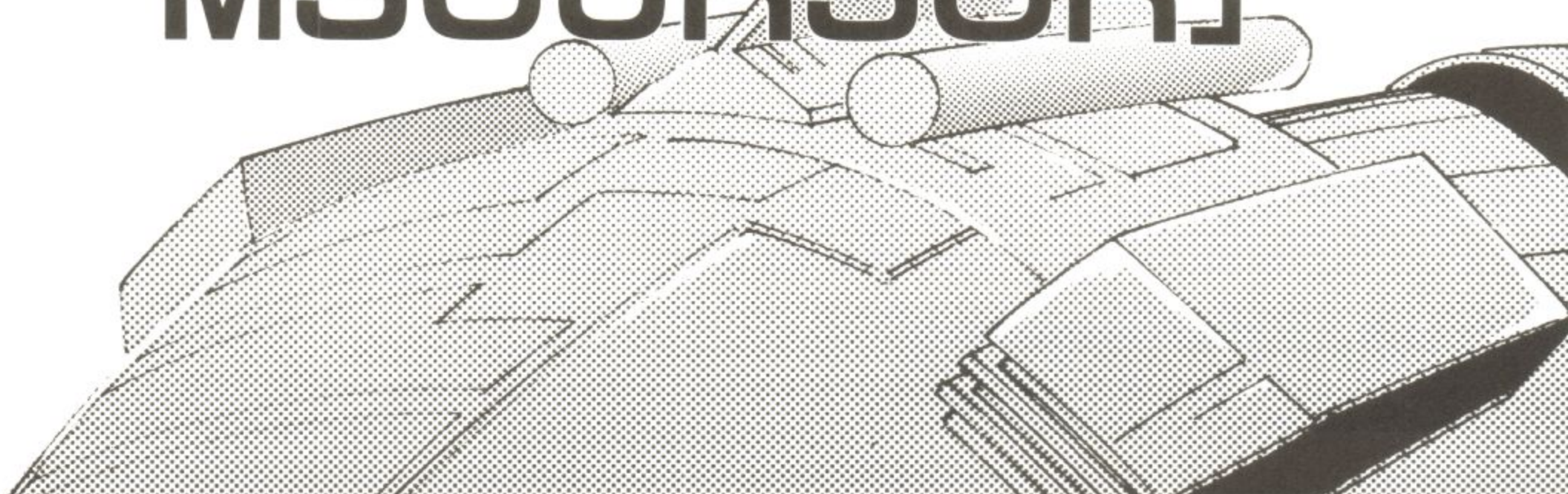
この関数を使ったサンプルプログラムは，本章の最後にあるTEXT_DRAW.BASにありますので参考にして下さい。しかし，このms_csetで定義することが出来るパターンは一つだけです。この関数を使用してマウスカーソルを作るのは，面倒な作業です。頑張ってオリジナルのカーソルを作ってみて下さい。

ワンポイントテクニック　テンキーでテレビの操作を

パソコンを使っている時に，突然テレビが見たくなった時に，X68000はあなたの味方です。秘密のキー（なにも秘密ではない）ですぐテレビ画面に早変わり。OPT.2＋テンキーでテレビ画面を操作出来ます。リモコンに付いている操作はすべて出来ます。キー操作は次の通りです。

《キー操作》

	テンキー	動作
[OPT.2]＋	0	消音
	,（カンマ）	標準音量
	.（ピリオド）	テレビ／パソコン切替
	1	1チャンネル
	2	2チャンネル
	3	3チャンネル
	4	4チャンネル
	5	5チャンネル
	6	6チャンネル
	7	7チャンネル
	8	8チャンネル
	9	9チャンネル
	／	10チャンネル
	*	11チャンネル
	－	12チャンネル
	[CLR	チャンネル表示ON/OFF
	＋	スーパーインポーズON/OFF

5-4 使用したいマウスカーソル番号を設定する関数「MSCURSOR」

仕様書４

関数名：mscursor（n）
引数の数と型：int n
戻り値：ナシ
内容：引数で指定された番号のマウスカーソルをセットします。

リスト４

```
**********************************************************
*                                                        *
*          BASIC EXTERNAL FUNCTION PROGRAM LIBRARY       *
*                                                        *
*                  月刊マイコン　別冊                    *
*                                                        *
*                昭和６２年７月２５日　制作              *
*                                                        *
*            PROGRAMED & Typed by　　深沢　幸三          *
*                                                        *
*      《 ＭＳＣＵＲＳＯＲ 》　（Ｖｅｒ１．０）          *
*                                                        *
**********************************************************


MS_SEL          equ     $007b


int_val         equ     $0002
```

```
int_ret         equ     $8000

adr_init        dc.l    return
adr_run         dc.l    return
adr_end         dc.l    return
adr_system      dc.l    return
adr_break       dc.l    return
adr_input       dc.l    return
adr_reserve1    dc.l    return
adr_reserve2    dc.l    return
adr_token       dc.l    token_table
adr_parameter   dc.l    param_adr_table
adr_exec        dc.l    exec_table
reserve         dc.l    0,0,0,0,0


token_table     dc.b    "mscursor",0,0,0

                .even

param_adr_table dc.l    param_table

param_table     dc.w    int_val,int_ret


                .even

exec_table      dc.l    mscursor

                .even

mscursor:       move.l  12(sp),d2
                move.w  d2,d1
                moveq   #MS_SEL,d0
```

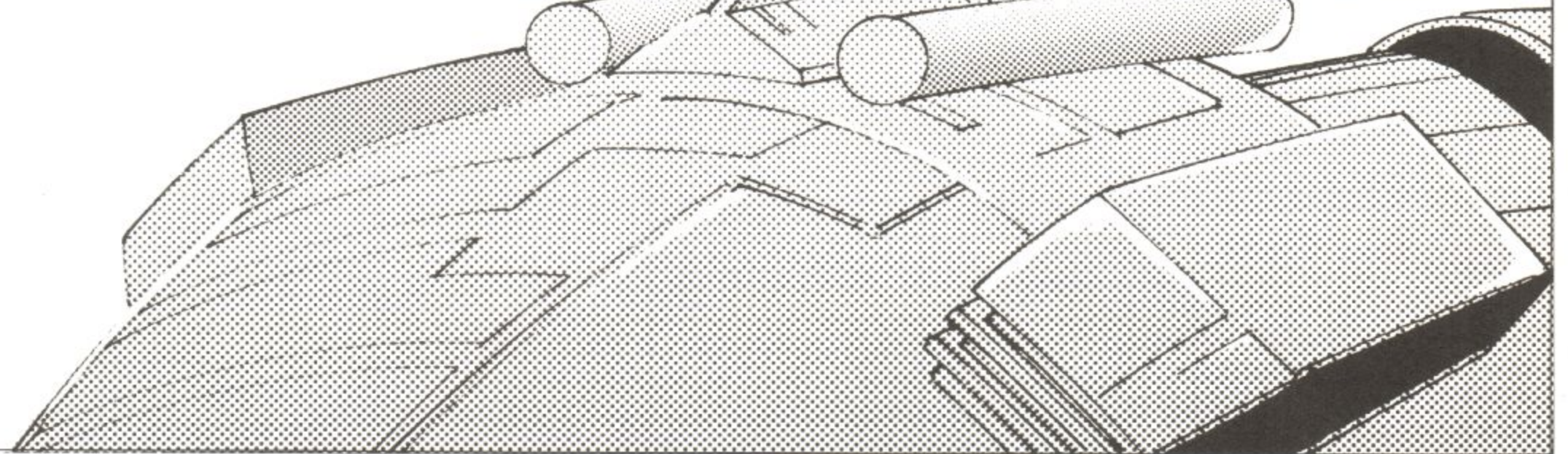
```
                trap    #15

                clr.l   d0
                lea.l   ret_param(pc),a0
                lea.l   msg_error(pc),a1

return          rts

                .even

ret_param       dc.w    0
                dc.l    0
                dc.l    1

msg_error       dc.b    "Error",0
```

この関数に使用されているIOCSコールは次の通りです。比較的に簡単なIOCSコールです。マウスカーソルの番号を指定レジスタのd1にワードサイズで設定してIOCSコール番号をd0にセットして，trap　＃15とするだけです。

IOCSコール番号：＄7b
IOCS名：MS_SEL

指定レジスタ：d1.L＝カーソル番号

この関数を使ったサンプルプログラムは，本章の最後にあるTEXT_DRAW.BASにありますので参考にして下さい。

5-5 複合関数「XLINE」「TLINE」「TCLS」

XLINEはグラフィック画面，XORモード（交点で色のXORをとる）で線を引きます。TLINEはテキスト画面に，ORモード（交点で色のorをとる）で線を引きます。TCLSはテキストのページ単位のクリアです。

この関数は，一つのファイルに三つの関数が入れてある複合関数です。

（1）XLINE

仕様書5

関数名：XLINE（x1，y1，x2，y2，c）
引数の数と型：int x1，x2，y1，y2，c
戻り値：ナシ
内容：cで指定した色（0～655色）で，x1，y1からx2，y2まで線を引きます。この関数は，512×512の65536色モードのみ有効です（他のモードでも使用出来ますが，動作については保証されていません）。

```
 10 int x1,x2,y1,y2,c,i
 20 for i=0 to 999
 30     x1=rand()/64
 40     y1=rand()/64
 50     x2=rand()/64
 60     y2=rand()/64
 70     c=rand()*2
 80     xline(x1,y1,x2,y2,c)
 90 next
100 end
```

第5－4図　XLINEのサンプルプログラム

このプログラムを実行する前に，ダイレクトモードでscreen，1，3，1，1を実行して下さい。xorでの描画は交点の色が論理演算でxorをとります。従って同じ色で同じ位置に線を引くと表示されていない状態になります。これは，グラフィック画面が1枚しかない時に有効で，重なった図形からある図形だけを消すような場合に指定された図形と交点を持つ図形の交点を消さなくて済みます。

X-BASICの乱数系は，初期化されないと同じ乱数を発生しますから，二度目に実行した時は一回目に書いた線を一本ずつ消していきます。そのため，一回目に書いた線を消さないように，プログラム中にはscreen文が書いてありません。

（2）TLINE

仕様書6

関数名：TLINE（x1，y1，x2，y2，pg）
引数の数と型：int x1，x2，y1，y2，pg
戻り値：ナシ
内容：pgで指定されたテキストページに，x1，y1，からx2，y2まで線を引きます。指定範囲は，768×512モードだけしか保証されていません。
　pg＝0……テキストページ1
　　　1……テキストページ2
　　　2……テキストページ3
　　　3……テキストページ4

この関数を使ったサンプルプログラムは，本章の最後にあるTEXT_DRAW.BASにありますので参考にして下さい。

このtline関数では，テキスト画面をあたかもグラフィック画面のように使用しています。これは，X68000のテキスト画面がフルビットマップ方式であるために実質的には，グラフィック画面のような使い方が出来るからです。このテキスト画面は4ページあり，0，1ページでは文字が，2，3ページではマウスが使用されています。従ってマウスを使用するようなプログラムでは，2，3ページの使用はさけた方がよいでしょう。

さて，テキスト画面をグラフィック画面のように使用して作った絵を消す方法ですが，次に紹介するtcls関数を使用して消去できますが，面白い消し方で，テキスト画面クリアのclsステートメントがありますが，これを使ってtlineで描いた絵を消すことが出来ます。グラフィックのように描いて，clsで消す。こんなことができるのもX68000の楽しみの一つです。

（3）TCLS

仕様書7

関数名：tcls（pg）

引数の数と型：int pg

戻り値：ナシ

内容：pgで指定されたテキストページをクリアします。

pg＝－1	全ページ
pg＝0	ページ0
pg＝1	ページ1
pg＝2	ページ2
pg＝3	ページ3

この関数を使ったサンプルプログラムは，本章の最後にあるTEXT_DRAW.BASにありますので参考にして下さい。

この関数は，さきほど紹介したtlineにより指定したテキストページに線を引くことが出来ましたが，このtcls関数は指定したテキストページだけを消去することが出来ます。

(1)～(3)の関数を一つのヘッダにまとめたものが**リスト5**です。

リスト5

```
*****************************************************************
*                                                               *
*         BASIC EXTERNAL FUNCTION PROGRAM LIBRARY               *
*                                                               *
*                   月刊マイコン　別冊                          *
*                                                               *
*               昭和６２年８月１１日　制作                      *
*                                                               *
*            PROGRAMED & Typed by   宮原　哲也                  *
*                                                               *
* 〈 XLINE  TLINE  TCLS 〉（Ver1.0）                            *
*                                                               *
*****************************************************************

*************** IOCS ﾈｰﾑ *****************************

_exit           equ             $ff00
_super          equ             $ff20
                 text
*************** ﾋｷｽｳ･ﾓﾄﾞﾘﾁ*******************************
```

```
int_val         equ             $0002
int_ret         equ             $8001

*************** ｲﾝﾌｫﾒｰｼｮﾝ・ﾃｰﾌﾞﾙ**********************

adr_init        dc.l            return
adr_run         dc.l            return
adr_end         dc.l            return
adr_system      dc.l            return
adr_break       dc.l            return
adr_input       dc.l            return
adr_reserve1    dc.l            return
adr_reserve2    dc.l            return
adr_token       dc.l            token_table
adr_parameter   dc.l            param_adr_table
adr_exec        dc.l            exec_table
reserve         dc.l            0,0,0,0,0

*************** ﾄｰｸﾝ・ﾃｰﾌﾞﾙ****************************

token_table     dc.b            "tline",0
                dc.b            "xline",0
                dc.b            "tcls",0

*************** ﾊﾟﾗﾒｰﾀ・ﾃｰﾌﾞﾙ**************************
                .even
param_adr_table dc.l            param_table_1
                dc.l            param_table_2
                dc.l            param_table_3

param_table_1   dc.w            int_val,int_val,int_val
                dc.w            int_val,int_val,int_ret

param_table_2   dc.w            int_val,int_val,int_val
```

```
                dc.w            int_val,int_val,int_ret

param_table_3   dc.w            int_val,int_ret

*************** ｼﾞｯｺｳｱﾄﾞﾚｽ･ﾃｰﾌﾞﾙ***********************
                .even
exec_table      dc.l            tline
                dc.l            xline
                dc.l            tcls

*************** TEXT LINE ******************************
                .even
tline:
                move.l          12(sp),X1
                move.l          22(sp),Y1
                move.l          32(sp),X2
                move.l          42(sp),Y2
                move.l          52(sp),COL

                clr.l           -(sp)
                dc.w            _super
                addq.l          #4,sp
                move.l          d0,sspbuf
                move.l          usp,a0
                move.l          a0,uspbuf

                bsr             line_1

                move.l          uspbuf,a0
                move.l          a0,usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp
```

```
                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

                rts

*************** TEXT LINE ﾎﾝﾀｲ****************************
                .even
line_1:
                move.l          X1,d0
                move.l          X2,d1
                move.l          Y1,d2
                move.l          Y2,d3
                move.l          d0,d6
                move.l          d2,d7
                sub.l           d0,d1
                sub.l           d2,d3
                cmp.l           #0,d1
                bgt.l           PLUSX_1
                beq.l           ZEROX_1
                move.l          #-1,SGNX
                muls            #-1,d1
                bra             CONT1_1
ZEROX_1         move.l          #0,SGNX
                bra             CONT1_1
PLUSX_1         move.l          #1,SGNX
CONT1_1         move.l          d1,ABSDLTX
                muls            #2,d1
                move.l          d1,ABSDLTX2
                move.l          ABSDLTX,d1
                cmp.l           #0,d3
                bgt.l           PLUSY_1
                beq.l           ZEROY_1
                move.l          #-1,SGNY
```

```
            muls        #-1,d3
            bra         CONT2_1
ZEROY_1     move.l      #0,SGNY
            bra         CONT2_1
PLUSY_1     move.l      #1,SGNY
CONT2_1     move.l      d3,ABSDLTY
            muls        #2,d3
            move.l      d3,ABSDLTY2
            move.l      ABSDLTY,d3
            cmp.l       d1,d3
            bgt.l       BIGY_1
BIGX_1      move.l      ABSDLTX,d4
            muls        #-1,d4
            clr.l       d5
LOOP1_1     move.l      d6,SX
            move.l      d7,SY

            bsr         TEXTPSET

            add.l       SGNX,d6
            add.l       ABSDLTY2,d4
            cmp.l       #0,d4
            blt.l       CONTBIGX_1
            add.l       SGNY,d7
            sub.l       ABSDLTX2,d4
CONTBIGX_1  addq.l      #1,d5
            cmp.l       d1,d5
            blt.l       LOOP1_1
            bra         CONT_1

BIGY_1      move.l      ABSDLTY,d4
            muls        #-1,d4
            clr.l       d5
LOOP2_1     move.l      d6,SX
```

```
                move.l          d7,SY
                bsr             TEXTPSET
                add.l           SGNY,d7
                add.l           ABSDLTX2,d4
                cmp.l           #0,d4
                blt.l           CONTBIGY_1
                add.l           SGNX,d6
                sub.l           ABSDLTY2,d4
CONTBIGY_1      addq.l          #1,d5
                cmp.l           d3,d5
                blt.l           LOOP2_1

CONT_1          rts

*************** TEXT PSET*********************************

TEXTPSET:
                move.l          d6,TEMPX
                move.l          d7,TEMPY
                mulu            #$80,d7
                divu            #16,d6
                mulu            #2,d6
                add.l           d6,d7
                move.l          d7,adrs1
                mulu            #8,d6
                move.l          TEMPX,d7
                sub.l           d6,d7
                move.l          #$8000,d6
                lsr.l           d7,d6
                move.l          COL,d7
                cmp.l           #0,d7
                beq             COLOR0
                cmp.l           #1,d7
                beq             COLOR1
```

```
                cmp.l           #2,d7
                beq             COLOR2
                bra             COLOR3

COLOR0          move.l          adrsl,d7
                add.l           #$E00000,d7
                move.l          d7,a4
                or.w            d6,(a4)
                move.l          TEMPY,d7
                move.l          TEMPX,d6
                rts

COLOR1          move.l          adrsl,d7
                add.l           #$E20000,d7
                move.l          d7,a4
                or.w            d6,(a4)
                move.l          TEMPY,d7
                move.l          TEMPX,d6
                rts

COLOR2          move.l          adrsl,d7
                add.l           #$E40000,d7
                move.l          d7,a4
                or.w            d6,(a4)
                move.l          TEMPY,d7
                move.l          TEMPX,d6
                rts

COLOR3          move.l          adrsl,d7
                add.l           #$E60000,d7
                move.l          d7,a4
                or.w            d6,(a4)
                move.l          TEMPY,d7
                move.l          TEMPX,d6
```

```
                rts

*************** XOR LINE***********************
                .even
xline:
                move.l          12(sp),X1
                move.l          22(sp),Y1
                move.l          32(sp),X2
                move.l          42(sp),Y2
                move.l          52(sp),COL

                clr.l           -(sp)
                dc.w            _super
                addq.l          #4,sp
                move.l          d0,sspbuf
                move.l          usp,a0
                move.l          a0,uspbuf

                bsr             line_2

                move.l          uspbuf,a0
                move.l          a0,usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp

                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

                rts

*************** XOR LINE ﾎﾝﾀｲ ***********************
line_2:
```

```
            move.l      X1,d0
            move.l      X2,d1
            move.l      Y1,d2
            move.l      Y2,d3
            move.l      d0,d6
            move.l      d2,d7
            sub.l       d0,d1
            sub.l       d2,d3
            cmp.l       #0,d1
            bgt.l       PLUSX_2
            beq.l       ZEROX_2
            move.l      #-1,SGNX
            muls        #-1,d1
            bra         CONT1_2
ZEROX_2     move.l      #0,SGNX
            bra         CONT1_2
PLUSX_2     move.l      #1,SGNX
CONT1_2     move.l      d1,ABSDLTX
            muls        #2,d1
            move.l      d1,ABSDLTX2
            move.l      ABSDLTX,d1
            cmp.l       #0,d3
            bgt.l       PLUSY_2
            beq.l       ZEROY_2
            move.l      #-1,SGNY
            muls        #-1,d3
            bra         CONT2_2
ZEROY_2     move.l      #0,SGNY
            bra         CONT2_2
PLUSY_2     move.l      #1,SGNY
CONT2_2     move.l      d3,ABSDLTY
            muls        #2,d3
            move.l      d3,ABSDLTY2
            move.l      ABSDLTY,d3
```

```
              cmp.l        d1,d3
              bgt.l        BIGY_2
BIGX_2        move.l       ABSDLTX,d4
              muls         #-1,d4
              clr.l        d5
LOOP1_2       move.l       d6,SX
              move.l       d7,SY
              bsr          XORPSET
              add.l        SGNX,d6
              add.l        ABSDLTY2,d4
              cmp.l        #0,d4
              blt.l        CONTBIGX_2
              add.l        SGNY,d7
              sub.l        ABSDLTX2,d4
CONTBIGX_2    addq.l       #1,d5
              cmp.l        d1,d5
              blt.l        LOOP1_2
              bra          CONT_2

BIGY_2        move.l       ABSDLTY,d4
              muls         #-1,d4
              clr.l        d5
LOOP2_2       move.l       d6,SX
              move.l       d7,SY
              bsr          XORPSET
              add.l        SGNY,d7
              add.l        ABSDLTX2,d4
              cmp.l        #0,d4
              blt.l        CONTBIGY_2
              add.l        SGNX,d6
              sub.l        ABSDLTY2,d4
CONTBIGY_2    addq.l       #1,d5
              cmp.l        d3,d5
              blt.l        LOOP2_2
```

```
CONT_2          rts

*************** XOR PSET ********************************
                .even
XORPSET:
                move.l          d6,TEMPX
                move.l          d7,TEMPY
                mulu            #1024,d7
                add.l           d6,d7
                add.l           d6,d7
                add.l           #$c00000,d7
                move.l          d7,a4
                move.l          COL,d7
                eor.l           d7,(a4)
                move.l          TEMPY,d7
                move.l          TEMPX,d6
                rts

*********************************************************
                .even
tcls:
                move.l          12(sp),COL

                clr.l           -(sp)
                dc.w            _super
                addq.l          #4,sp
                move.l          d0,sspbuf
                move.l          usp,a0
                move.l          a0,uspbuf

                move.l          COL,d0
                cmp.l           #0,d0
                blt             clsall
                cmp.l           #0,d0
```

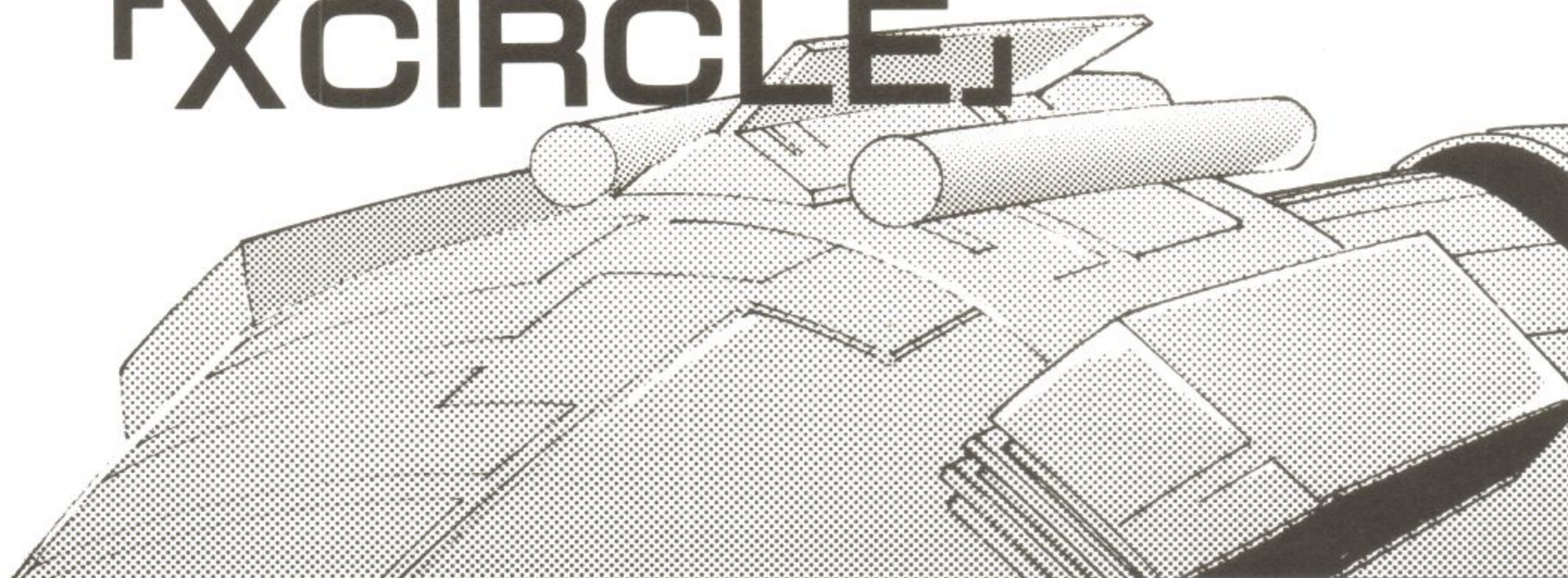
```
                beq             cls0
                cmp.l           #1,d0
                beq             cls1
                cmp.l           #2,d0
                beq             cls2
                cmp.l           #3,d0
                beq             cls3


                move.l          uspbuf,a0
                move.l          a0,usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp

                move.l          #-1,d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

                rts

**************************************************************

clsall          move.l          #$e00000,a0
                move.l          #$e20000,a1
                move.l          #$e40000,a2
                move.l          #$e60000,a3
                move.l          #0,d0
transall        move.w          d0,(a0)
                move.w          d0,(a1)
                move.w          d0,(a2)
                move.w          d0,(a3)
                addq.l          #2,a0
                addq.l          #2,a1
```

```
                addq.l      #2,a2
                addq.l      #2,a3
                cmp.l       #$e20000,a0
                bne         transall

                bra         return_c

cls0            move.l      #$e00000,a0
                move.l      #0,d0
trans0          move.w      d0,(a0)
                addq.l      #2,a0
                cmp.l       #$e20000,a0
                bne         trans0

                bra         return_c

cls1            move.l      #$e20000,a0
                move.l      #0,d0
trans1          move.w      d0,(a0)
                addq.l      #2,a0
                cmp.l       #$e40000,a0
                bne         trans1

                bra         return_c

cls2            move.l      #$e40000,a0
                move.l      #0,d0
trans2          move.w      d0,(a0)
                addq.l      #2,a0
                cmp.l       #$e60000,a0
                bne         trans2

                bra         return_c
```

```
cls3            move.l          #$e60000,a0
                move.l          #0,d0
trans3          move.w          d0,(a0)
                addq.l          #2,a0
                cmp.l           #$e80000,a0
                bne             trans3


return_c        move.l          uspbuf,a0
                move.l          a0,usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp

                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

return          rts


*************** ﾘﾀｰﾝ ﾊﾟﾗﾒｰﾀ ******************************

                .even
ret_param       dc.w            0
                dc.l            0
                dc.l            1

msg_error       dc.b            "外部関数エラーです",0

sspbuf          dc.l            0

uspbuf          dc.l            0
```

```
*************** ﾜｰｸ ｴﾘｱ ***********************************


adrs1           dc.l            0
adrs2           dc.l            0
TEMPX           dc.l            0
TEMPY           dc.l            0
X1              dc.l            0
Y1              dc.l            0
X2              dc.l            0
Y2              dc.l            0
SX              dc.l            0
SY              dc.l            0
COL             dc.l            0
SGNX            dc.l            0
SGNY            dc.l            0
ABSDLTX         dc.l            0
ABSDLTY         dc.l            0
ABSDLTX2        dc.l            0
ABSDLTY2        dc.l            0


**********************************************************
                .end
```

この関数では，IOCSコールを使用していません。ここでは，基本概念として整数での処理によるライン処理の考え方であるDDA法を使っています。DDA法と言うのは，基本的に整数で直線の方程式を近似的に解くもので，グラフィック関係の書籍ではよく扱われている技法です。ともかく，マシン語で実数計算をするのが非常に難しいため，こうした整数で処理を行うものについての知識を持っていると便利です。

5-6 XORタイプのCIRCLE「XCIRCLE」

仕様書８

書式：xcircle（x１，y１，r，c，h）
引き数の型と数：int　x１，y１，r，c，h
戻り値：ナシ
内容：x1，y1を中心とし，半径rの円をcで指定した色で描きます。hは偏平率で，48にした時に真円となります。この関数では，512×512の65536色モードしか保証されていません。

リスト６

```
**************************************************************
*                                                            *
*          BASIC EXTERNAL FUNCTION PROGRAM LIBRARY           *
*                                                            *
*                   月刊マイコン　別冊                       *
*                                                            *
*                  昭和６２年７月１６日　制作                *
*                                                            *
*             PROGRAMED & Typed by   宮原　哲也              *
*                                                            *
*〈〈 XOR  CIRCLE 〉〉     （Ver1.0）                        *
*                                                            *
**************************************************************


_exit           equ             $ff00
                .text
************************************************

int_val         equ             $0002
int_ret         equ             $8001
```

```
**********************************************

adr_init        dc.l        return
adr_run         dc.l        return
adr_end         dc.l        return
adr_system      dc.l        return
adr_break       dc.l        return
adr_input       dc.l        return
adr_reserve1    dc.l        return
adr_reserve2    dc.l        return
adr_token       dc.l        token_table
adr_parameter   dc.l        param_adr_table
adr_exec        dc.l        exec_table
reserve         dc.l        0,0,0,0,0

**********************************************


token_table     dc.b        "xcircle",0,0,0

**********************************************
                .even

param_adr_table dc.l        param_table

param_table     dc.w        int_val,int_val,int_val
                dc.w        int_val,int_val,int_ret

**********************************************
                .even
exec_table      dc.l        xcircle

**********************************************
                .even
xcircle:
                move.l      12(sp),X0
                move.l      22(sp),Y0
                move.l      32(sp),RR
                move.l      42(sp),COL
                move.l      52(sp),HSV

                clr.l       -(sp)
                dc.w        _super
                addq.l      #4,sp
                move.l      d0,sspbuf
```

```
                move.l          usp,a0
                move.l          a0.uspbuf

                bsr             CIRCLE
                move.l          uspbuf,a0
                move.l          a0.usp
                move.l          sspbuf,-(sp)
                dc.w            _super
                addq.l          #4,sp
                clr.l           d0
                lea.l           ret_param(pc),a0
                lea.l           msg_error(pc),a1

return          rts

                .even
ret_param       dc.w            0
                dc.l            0
                dc.l            1

msg_error       dc.b            "外部関数エラーです",0

sspbuf          dc.l            0

uspbuf          dc.l            0

************************************************************
                .even
CIRCLE:
                move.l          RR,d2           ;
                muls            #64,d2          ;
                move.l          d2,X            ;X=R*64
                clr.l           d2              ;
                move.l          d2,Y            ;Y=0
loop            move.l          Y,d2            ;
                move.l          RR,d3           ;
                divs            d3,d2           ;Y/R
                andi.l          #$ffff,d2
                move.l          X,d3            ;
                sub.l           d2,d3           ;X-Y/R
                move.l          d3,XN           ;XN=X-Y/R
                move.l          RR,d2           ;
                divs            d2,d3           ;XN/R
                andi.l          #$ffff,d3
```

```
                move.l      Y,d2            ;
                add.l       d2,d3           ;XN/R+Y
                move.l      d3,YN           ;YN=XN/R+Y
                move.l      XN,X            ;X=XN
                move.l      YN,Y            ;Y=YN
                move.l      XN,d0           ;
                move.l      HSV,d1          ;
                muls        d1,d0           ;XN*HSV
                divs        #64,d0          ;(XN*HSV)/64
                andi.l      #$ffff,d0
                move.l      X0,d1           ;
                muls        #64,d1          ;X0*64
                add.l       d1,d0           ;X0*64+(XN*HSV)/64
                divs        #64,d0          ;(X0*64+(XN*HSV)/64)
/64
                andi.l      #$ffff,d0
                move.w      d0,SX0          ;SX0=(X0*64+(XN*HSV)
/64)/64
                move.l      Y0,d1           ;
                move.l      YN,d2           ;
                muls        #64,d1          ;Y0*64
                sub.l       d2,d1           ;Y0*64-YN
                divs        #64,d1          ;(Y0*64-YN)/64
                andi.l      #$ffff,d1
                addq.l      #1,d1           ;(Y0*64-YN)/64+1
                move.w      d1,SY0          ;SY0=(Y0*64-YN)/64+1
                move.l      X0,d0           ;
                muls        #2,d0           ;2*X0
                move.w      SX0,d1          ;
                sub.l       d1,d0           ;2*X0-SX0
                move.w      d0,SX1          ;SX1=X0*2-SX0
                move.l      Y0,d0           ;
                muls        #2,d0           ;2*Y0
                move.w      SY0,d1          ;
                sub.l       d1,d0           ;
                move.w      d0,SY1          ;SY1=Y0*2-SY0
                bsr         XORPSET1        ;
                move.w      SX0,d0          ;
                move.l      X0,d1           ;
                sub.w       d1,d0           ;
                cmp.w       #0,d0           ;
                bgt         loop            ;

                rts
```

```
*************************************************************************
            .even
XORPSET1:
            move.w      SX0,d6
            move.w      SY0,d7
            mulu        #1024,d7
            add.l       d6,d7
            add.l       d6,d7
            add.l       #$c00000,d7
            move.l      d7,a4
            move.l      COL,d7
            or.w        d7,(a4)

            move.w      SX1,d6
            move.w      SY0,d7
            mulu        #1024,d7
            add.l       d6,d7
            add.l       d6,d7
            add.l       #$c00000,d7
            move.l      d7,a4
            move.l      COL,d7
            or.w        d7,(a4)

            move.w      SX0,d6
            move.w      SY1,d7
            mulu        #1024,d7
            add.l       d6,d7
            add.l       d6,d7
            add.l       #$c00000,d7
            move.l      d7,a4
            move.l      COL,d7
            or.w        d7,(a4)


            move.w      SX1,d6
            move.w      SY1,d7
            mulu        #1024,d7
            add.l       d6,d7
            add.l       d6,d7
            add.l       #$c00000,d7
            move.l      d7,a4
            move.l      COL,d7
            or.w        d7,(a4)
            rts
```

```
X               dc.l        0
Y               dc.l        0
X0              dc.l        0
Y0              dc.l        0
RR              dc.l        0
SX0             dc.l        0
SY0             dc.l        0
SX1             dc.l        0
SY1             dc.l        0
COL             dc.l        0
XN              dc.l        0
YN              dc.l        0
HSV             dc.l        0
*
                .end
```

この関数を使用したX-BASICのサンプルプログラムは，**第5－5図**の通りです。

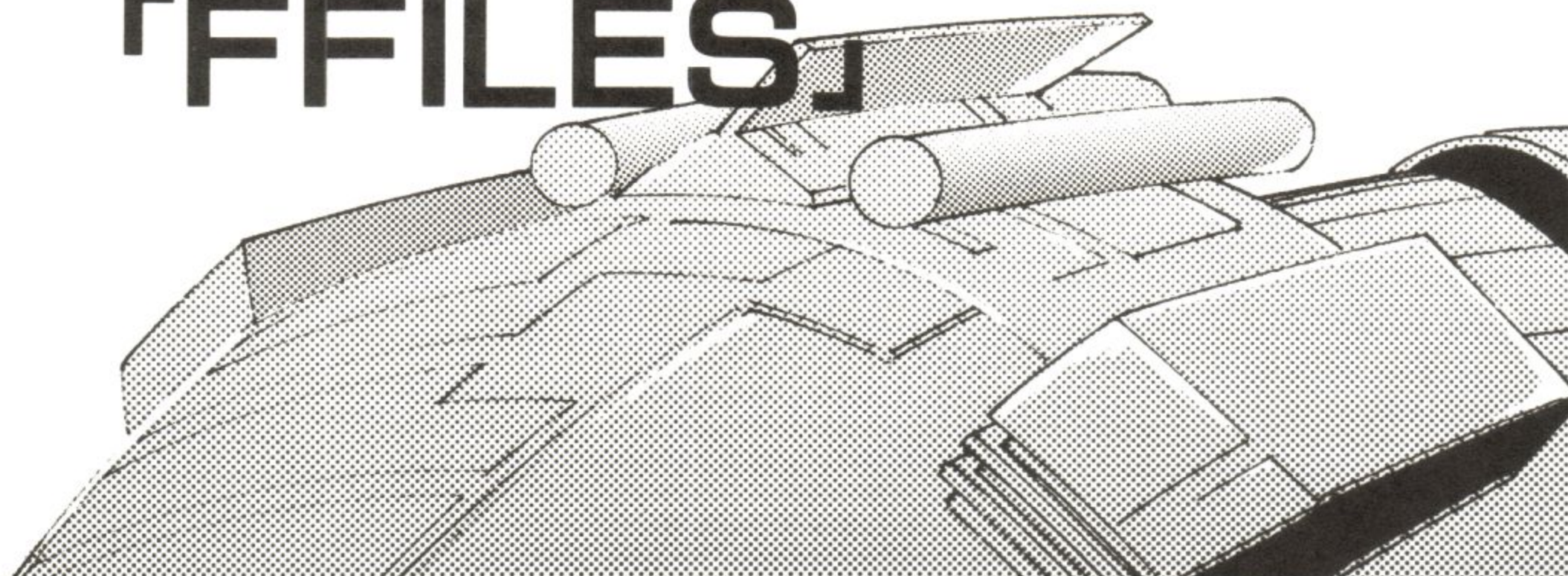
```
 10 int x,y,c,r,i
 20 for i=0 to 999
 30      x=rand()/128
 40      y=rand()/128
 50      r=rand()/327
 60      c=rand()*2
 70 if r=0 or x-r<0 or y-r<0 or x+r>511 or y+r>511 then continue
 80 xcircle(x,y,r,c,48)
 90 next
100 end
```

第5－5図　XCIRCLEのサンプルプログラム

このプログラムは，終了した後にもう一回実行するとxorの面白さがわかります。乱数系を初期化していないので，もう一度実行した時も同じ乱数が発生するので，忠実に一回目のサークルをトレースしていきます。ただし，このプログラムでは，xlineの時と同じく二回目を実行する時にグラフィック画面を消去しないようにscreen命令を使用していません。一回目の実行の前に，ダイレクトモードで，

　　　　screen　1，3，1，1↵

としてから実行して下さい。実行結果は，xlineのものと同じになると思います。

5-7 検索機能付FILES関数「FFILES」

仕様書９

関数名：n=ffiles（x，f，st）
引数の数と型：int x
　　　　　　　str f
　　　　　　　str型の配列　st
戻り値：int n
内容：配列宣言されているstとその配列の最大数xと検索文字列fとで，ファイルを検索し該当ファイルを引数の文字配列に格納して該当ファイル数を戻り値として返します。

リスト７

```
****************************************************************
*                                                              *
*          BASIC EXTERNAL FUNCTION PROGRAM LIBRARY             *
*                                                              *
*                     月刊マイコン　別冊                       *
*                                                              *
*                 昭和６２年８月１８日　制作                   *
*                                                              *
*            PROGRAMED & Typed by　　宮原　哲也                *
*                                                              *
*       《 ＦＦＩＬＥＳ 》        （Ｖｅｒ１．０）             *
*                                                              *
****************************************************************

_files          equ     $ff4e
```

```
_nfiles         equ     $ff4f

**************** Parameter ID ***************************

int_val         equ     $0002

str_val         equ     $0008           ;文字列の引数用

str_dim         equ     $0038           ;文字配列の引数用

int_ret         equ     $8001           ;整数の戻り値用

***************** Information Table **********************
                .even

a_init          dc.l    return
a_run           dc.l    return
a_end           dc.l    return
a_sys           dc.l    return
a_brk           dc.l    return
a_inp           dc.l    return
a_rsrv1         dc.l    return
a_rsrv2         dc.l    return
a_token         dc.l    token_tbl
a_parm          dc.l    parm_a_tbl
a_exec          dc.l    exec_tbl
reserve         dc.l    0,0,0,0,0

***************** Token Table *****************************
                .even

token_tbl       dc.b    "ffiles",0,0

***************** Parameter Table *************************
```

```
                .even

parm_a_tbl      dc.l    parm_tbl

parm_tbl        dc.w    int_val,str_val,str_dim,int_ret

**************** Exec Adr. Table ***********************
                .even

exec_tbl        dc.l    ffiles

**************** Function Main *************************
                .even
ffiles:
                move.l  12(sp),d1
                movea.l 22(sp),a2
                movea.l 32(sp),a1
                lea.l   10(a1),a1

                clr.l   -(sp)
                dc.w    _super
                addq.l  #4,sp
                move.l  d0,sspbuf
                move.l  usp,a0
                move.l  a0,uspbuf

                lea.l   atr,a3

next_byte       move.b  (a2)+,(a3)+
                move.b  (a2),d0
                cmp.b   #0,d0
                bne     next_byte
                move.b  (a2),(a3)
```

```
                clr.l   d2

                move.w  d1,-(sp)
                pea     atr
                pea     filebuf
                dc.w    _files
                lea     10(sp),sp

                cmp.l   #0,d0
                bmi     comeback

                addq.l  #1,d2
                movea.l a1,a3
next_read       lea     name,a2
                movea.l a3,a1

next_fname      move.b  (a2)+,(a1)+
                move.b  (a2),d1
                cmp.b   #0,d1
                bne     next_fname

                move.b  (a2),(a1)

                pea     filebuf
                dc.w    _nfiles
                addq.l  #4,sp

                cmp.l   #0,d0
                bmi     comeback

                addq.l  #1,d2
                adda.l  #33,a3
                bra     next_read
```

```
comeback        move.l  d2,answer+6

                move.l  uspbuf,a0
                move.l  a0,usp
                move.l  sspbuf,-(sp)
                dc.w    _super
                addq.l  #4,sp

continue        clr.l   d0
                lea     answer(pc),a0
                lea     msgerror,a1

return          rts

                .even
sspbuf          dc.l    0
uspbuf          dc.l    0

answer          dc.w    0
                dc.l    0
                ds.l    1

msgerror        dc.b    "error",0

atr             ds.b    23

                .even
filebuf         dc.b    0
                dc.b    0
                dc.w    0
                dc.w    0
                dc.w    0
                dc.w    0
                ds.b    8
```

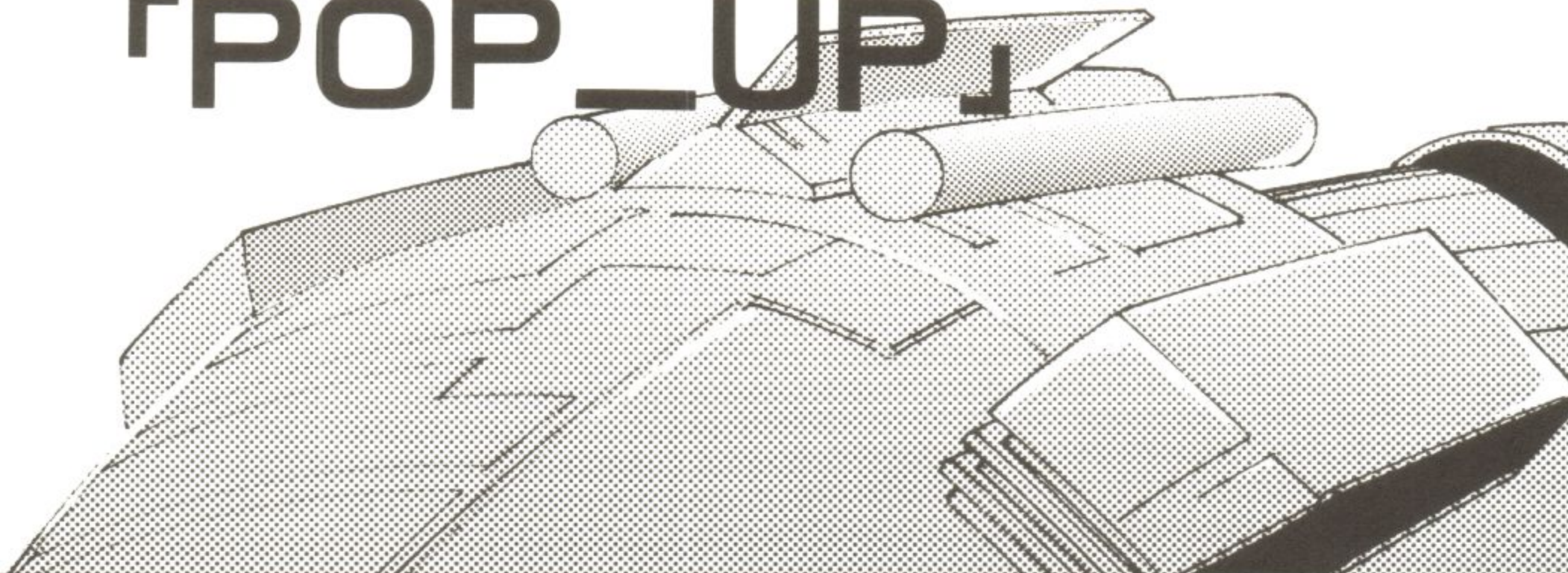
```
                ds.b    3
                dc.b    0
                dc.w    0
                dc.w    0
                dc.l    0
name            ds.b    23
                end
```

このプログラムでは，IOCSコールを使用していません。ファンクションコールにより基本的なファイル操作をしていますので，Human68kマニュアル巻末のファンクションコール一覧を参照して下さい。ffiles関数を使用したX-BASICのサンプルプログラムは，**第5－6図**の通りです。

```
10 str s
20 int x=50,n,i,m
30 input"input file name:",s
40 dim str f(50)
50 n=ffiles(x,s,f)
60 for i=0 to n
70     print f(i)
80 next
90 end
```

第5－6図　FFILESのサンプルプログラム

このプログラムを実行すると，画面に検索したいファイルネームを聞いてきますから，検索したいファイルネームを入力して下さい。戻り値のnが0の場合は，そのファイルはそのディレクトリ内にはない。または，そのフロッピー内にないことになります。指定するファイルネームは，ワイルドカードが使用出来るため，＊．＊とすればファイルすべてが，＊．fncとすれば＊＊＊．fncと言うファイルすべてが配列にセットされて戻ります。プログラムを作り，データ管理をするときにどんなファイルが存在するかをX-BASICのプログラム中に関数として持つことは非常にプログラムを簡略化させてくれます。

5-8 ポップアップメニュー関数「POP_UP」

仕様書10

関数名：s＝pop_up（st，n，x，y）

引数の数と型：str型の配列名　st

　　　　　　　int　n，x，y

戻り値：int　s

内容：str型の配列変数に格納されているメニュー項目を最初からn個だけx，yで与えられた座標に表示し，マウスを使用してその項目を示し左のマウスボタンを押すことにより，メニューをエントリーします。選ばれたメニューは，番号で戻り値として返り，枠の外でエントリーすると0が返ります。

この関数で使用したIOCSコールは次の通りです。また枠をテキスト画面に表示させるため，tline（　）の関数の一部を利用しています。

IOCSコール番号：$10……画面の表示モードを設定／現在の表示モードの読み取りをする
$15……テキストVRAMの指定をします
$1a……テキスト画面から指定範囲のパターンデータを読み込みます
$1b……テキスト画面から指定範囲のパターンデータを書き込みます
$21……テキスト画面に指定した文字列を書き込みます
$22……テキスト画面の表示属性をセットします
$23……カーソルを指定位置に移動させます
$71……マウスカーソルの表示をさせる
$74……マウスの移動量・ボタンのON／OFFの状態を読み取る
$75……マウスカーソルの位置を調べる
$76……マウスカーソルの位置を指定する

リスト8

```
*********************************************************
*                                                       *
*       BASIC EXTERNAL FUNCTION PROGRAM LIBRARY         *
*                                                       *
*              月刊マイコン　別冊                       *
*                                                       *
*           昭和６２年８月１１日　制作                  *
*                                                       *
*        PROGRAMED & Typed by   宮原　哲也              *
*                                                       *
* 〈〈 ＰＯＰ　ＵＰ　ＭＥＮＵ 〉〉　（Ｖｅｒ１．０）   *
*                                                       *
*********************************************************

**************** Parameter ID ****************************

int_val         equ     $0002           ;整数の引数用

str_dim         equ     $0038           ;文字配列の引数用

int_ret         equ     $8001           ;整数の戻り値用

***************** Information Table ***********************************
                .even

a_init          dc.l    return
a_run           dc.l    return
a_end           dc.l    return
a_sys           dc.l    return
a_brk           dc.l    return
a_inp           dc.l    return
a_rsrv1         dc.l    return
a_rsrv2         dc.l    return
a_token         dc.l    token_tbl
a_parm          dc.l    parm_a_tbl
a_exec          dc.l    exec_tbl
reserve         dc.l    0,0,0,0,0


***************** Token Table *****************************************
                .even

token_tbl       dc.b    "pop_up",0,0,0
```

```
***************** Parameter Table ***************************************
                .even

parm_a_tbl      dc.l    parm_tbl

parm_tbl        dc.w    str_dim,int_val,int_val,int_val
                dc.w    int_ret

***************** Exec Adr. Table ***************************************
                .even

exec_tbl        dc.l    pop_up

***************** Function Main *****************************************
pop_up:
                move.l  #0,max_len                  ;ワークエリアの初期化
                movea.l 12(sp),a0                   ;引数1
                lea.l   10(a0),a0                   ;引数1　文字列の
                move.l  a0,str_pointer              ;ポインターアドレス
                move.l  str_pointer,pointer
                move.l  22(sp),d0                   ;引数2　文字列の数
                move.l  d0,number
                move.l  32(sp),d1                   ;引数3　X座標
                move.l  d1,ms_x
                move.l  42(sp),d1                   ;引数4　Y座標
                move.l  d1,ms_y
                move.l  #33,d1                      ;文字列のオフセット
                cmp.l   #1,d0                       ;引数3が0か1の時
                ble     abort                       ;の処理
count_len       move.l  #-1,d2                      ;文字列の最大文字数
count_char      addq.l  #1,d2                       ;の計算
                tst.b   (a0)+
                bne     count_char
                move.l  max_len,d3
                cmp.l   d3,d2
                blt     no_opel
                move.l  d2,max_len
no_opel         subq.l  #1,d0
                beq     check_adr
                move.l  pointer,a0
                add.l   d1,a0
                move.l  a0,pointer
                bra     count_len
```

```
check_adr       move.w  #-1,d1          ;表示モードを
                moveq   #$10,d0         ;調べる
                trap    #15
                cmp.l   #16,d0
                beq     dot768
                lsr.l   #2,d0
                bcs     dot256
dot512          move.l  #511,max_x      ;表示モードにより
                move.l  #511,max_y      ;表示範囲を変更する
                bra     dot_check

dot256          move.l  #255,max_x
                move.l  #255,max_y
                bra     dot_check

dot768          move.l  #767,max_x
                move.l  #511,max_y

dot_check       move.l  ms_x,d1         ;表示可能範囲かどうか
                move.l  ms_y,d2         ;調べる
                move.l  number,d3
                move.l  max_len,d4
                divu    #8,d1
                divu    #16,d2
                ext.l   d1
                ext.l   d2
                mulu    #8,d1
                mulu    #16,d2
                mulu    #16,d3
                mulu    #8,d4

                cmp.l   #0,d1           ;表示範囲外なら
                beq     ofset_adr_xs    ;オフセットをかける
                cmp.l   #0,d2
                beq     ofset_adr_ys

                subq.l  #2,d1           ;メニューボックス用の
                move.l  d1,xs           ;4点の座標を計算する

                subq.l  #1,d2
                move.l  d2,ys

                add.l   d3,d2
                addq.l  #1,d2
```

```
                move.l  d2,ye

                add.l   d1,d4
                add.l   #3,d4
                move.l  d4,xe

                move.l  xe,d1
                cmp.l   max_x,d1
                bgt     ofset_adr_xe

                move.l  ye,d1
                cmp.l   max_y,d1
                bgt     ofset_adr_ye

menu_box        move.l  ms_y,d2             ;X座標、Y座標に
                divu    #16,d2              ;メニュー用の枠を
                ext.l   d2                  ;を作る
                move.l  ms_x,d1
                divu    #8,d1
                ext.l   d1
                move.l  d1,x_pos
                move.l  d2,y_pos

                moveq   #2,d1               ;ボックスを書く前に
                moveq   #$15,d0             ;ボックスの大きさだけ
                trap    #15                 ;テキストを保護する

                move.l  xe,d0               ;テキストページ2の保護
                sub.l   xs,d0
                addq.l  #3,d0
                move.w  d0,vram_buf2
                move.w  d0,wxe2
                move.l  xs,d0
                move.w  d0,wxs2

                move.l  ye,d0
                sub.l   ys,d0
                addq.l  #3,d0
                move.w  d0,vram_buf2+2
                move.w  d0,wye2
                move.l  ys,d0
                move.w  d0,wys2

                move.w  wxs2,d1
```

```
        move.w  wys2,d2
        lea.l   vram_buf2,a1
        moveq   #$1a,d0
        trap    #15

        moveq   #$1,d1                  ;テキストページ１の保護
        moveq   #$15,d0
        trap    #15

        move.l  xe,d0
        sub.l   xs,d0
        addq.l  #3,d0
        move.w  d0,vram_buf1
        move.w  d0,wxe1
        move.l  xs,d0
        move.w  d0,wxs1

        move.l  ye,d0
        sub.l   ys,d0
        addq.l  #3,d0
        move.w  d0,vram_buf1+2
        move.w  d0,wye1
        move.l  ys,d0
        move.w  d0,wys1

        move.w  wxs1,d1
        move.w  wys1,d2
        lea.l   vram_buf1,a1
        moveq   #$1a,d0
        trap    #15

        bsr     boxline                 ;ボックスの表示（影つき）

        move.l  #1,answer               ;答えの初期値

        bsr     print_menu              ;最初のメニュー表示

        moveq   #$71,d0                 ;マウスカーソルの
        trap    #15                     ;表示

        move.l  ms_x,d0                 ;マウスカーソルの
        lsl.l   #8,d0                   ;最初の位置のセット
        lsl.l   #8,d0
        andi.l  #$ffff0000,d0
```

```
                move.l  ms_y,d1
                ext.l   d1
                add.l   d0,d1

                moveq   #$76,d0
                trap    #15

ms_loop         moveq   #$75,d0                 ;マウス座標の読み込み
                trap    #15
                move.l  d0,d1
                andi.l  #$ffff,d1
                move.l  d1,mous_y
                lsr.l   #8,d0
                lsr.l   #8,d0
                andi.l  #$ffff,d0
                move.l  d0,mous_x

                move.l  xs,d1                   ;マウスカーソルが
                cmp.l   d0,d1                   ;メニュー内にいるか
                bhi     out_side                ;どうかの判定
                move.l  xe,d1
                cmp.l   d0,d1
                bcs     out_side
                move.l  mous_y,d0
                move.l  ys,d1
                cmp.l   d0,d1
                bhi     out_side
                move.l  ye.d1
                cmp.l   d0.d1
                bcs     out_side
                bra     in_side

out_side        move.l  answer,d1               ;メニュー外の処理
                beq     mous_btn
                move.l  #0,answer
                bsr     print_menu
                bra     mous_btn

in_side         move.l  ys,d1                   ;メニュー内の処理
                sub.l   d1,d0
                divu    #16,d0
                addq.l  #1,d0
                ext.l   d0
```

```
                move.l  answer,d1               ;答えを記憶する
                cmp.l   d1,d0
                beq     mous_btn
                move.l  d0,answer
                bsr     print_menu

mous_btn        moveq   #$74,d0                 ;マウスのボタンの
                trap    #15                     ;ON/OFF判定
                andi.l  #$0000ff00,d0
                beq     ms_loop                 ;OFFならループ

                move.l  answer,d0               ;ONの時はこちら
                ext.l   d0

                move.l  d0,ret_param+6          ;戻り値のセット

                moveq   #2,d1                   ;テキストページ2の復帰
                moveq   #$15,d0
                trap    #15

                move.w  wxe2,vram_buf2
                move.w  wye2,vram_buf2+2
                move.w  wxs2,d1
                move.w  wys2,d2
                lea.l   vram_buf2,a1
                moveq   #$1b,d0
                trap    #15

                moveq   #1,d1                   ;テキストページ1の復帰
                moveq   #$15,d0
                trap    #15

                move.w  wxe1,vram_buf1
                move.w  wye1,vram_buf1+2
                move.w  wxs1,d1
                move.w  wys1,d2
                lea.l   vram_buf1,a1
                moveq   #$1b,d0
                trap    #15

                clr.l   d0                      ;正常終了時の処理
                lea.l   ret_param(pc),a0        ;戻り値のアドレスをセットする

error           lea.l   msg_error(pc),a1        ;エラー時のメッセージアドレス
```

```
return          rts                                 ;関数の終了

abort           move.l  #-1,d0                      ;引数2が0だったら

                bra     error                       ;エラーとする

ofset_adr_xs    move.l  #8,ms_x                     ;表示範囲外の処理
                bra     check_adr                   ;ルーチン（0以下の処理）

ofset_adr_ys    move.l  #16,ms_y
                bra     check_adr

ofset_adr_xe    move.l  max_x,d0                    ;表示範囲外の処理
                move.l  xe.d1                       ;ルーチン
                sub.l   d0,d1                       ;表示範囲より大きい
                move.l  ms_x,d0                     ;場合の処理
                sub.l   d1.d0
                sub.l   #8,d0
                move.l  d0.ms_x
                bra     check_adr

ofset_adr_ye    move.l  max_y,d0
                move.l  ye,d1
                sub.l   d0.d1
                move.l  ms_y,d0
                sub.l   d1,d0
                sub.l   #32,d0
                move.l  d0,ms_y
                bra     check_adr

*////////////// PRINT MENU /////////////////////////////////////////////////

print_menu      move.l  str_pointer,pointer         ;文字列を表示開始位置から
                move.l  #1,d4                       ;配列を立てに積み重ね表示
                move.l  answer,d5                   ;してカーソル位置だけ
                move.l  x_pos,temp_x_pos            ;反転する
                move.l  y_pos,temp_y_pos
                move.l  number.d3

next_menu       cmp.l   d4,d5
                bne     normal
                move.l  #11,d1
                bra     color_set
```

```
normal          move.l  #3,d1

color_set       moveq   #$22,d0                 ;表示色のセット
                trap    #15

                move.l  temp_x_pos,d1
                move.l  temp_y_pos,d2

                moveq   #$23,d0                 ;LOCATE X, Y
                trap    #15
                move.l  max_len,d6

nextspc         lea.l   spc_code,a1             ;PRINT spc$(max_lex)
                moveq   #$21,d0
                trap    #15
                subq.l  #1,d6
                bne     nextspc

                move.l  temp_x_pos,d1
                move.l  temp_y_pos,d2

                moveq   #$23,d0                 ;LOCATE X, Y
                trap    #15

                move.l  pointer,a1              ;PRINT strings
                moveq   #$21,d0
                trap    #15
                subq.l  #1,d3                   ;指定配列数かどうか
                beq     print_menu_exit         ;のチェック
                move.l  pointer,a1
                add.l   #33,a1
                move.l  a1,pointer
                move.l  temp_y_pos,d2
                addq.l  #1,d2
                move.l  d2,temp_y_pos
                addq.l  #1,d4
                bra     next_menu

print_menu_exit move.l  #3,d1                   ;表示ルーチンの終了
                moveq   #$22,d0
                trap    #15

                rts
```

```
*////////////// TEXT BOX LINE /////////////////////////

boxline         move.l  xs,x1           ;ボックスの表示
                move.l  ys,y1           ;サブルーチン
                move.l  xe,x2           ;上の横線
                move.l  ys,y2

                bsr     txtline

                move.l  xs,x1           ;下の横線
                move.l  ye,y1
                move.l  xe,x2
                move.l  ye,y2

                bsr     txtline

                addq.l  #2,x1           ;下の横線の影
                addq.l  #2,x2
                addq.l  #2,y1
                addq.l  #2,y2

                bsr     txtline

                move.l  xs,x1           ;左の縦線
                move.l  xs,x2
                move.l  ys,y1
                move.l  ye,y2

                bsr     txtline

                move.l  xe,x1           ;右の縦線
                move.l  xe,x2
                move.l  ys,y1
                move.l  ye,y2

                bsr     txtline

                addq.l  #2,x1           ;右の縦線の影
                addq.l  #2,x2
                addq.l  #2,y1
                addq.l  #2,y2

                bsr     txtline
```

```
		rts


*////////////// TEXT Line routine ////////////////////////////////////////

_exit		equ		$ff00
_super		equ		$ff20

txtline:
		clr.l		-(sp)			;LINE ROUTINE
		dc.w		_super			;DDA法LINEルーチン
		addq.l		#4,sp
		move.l		d0,sspbuf
		move.l		usp,a0
		move.l		a0,uspbuf

		bsr		tline

		move.l		uspbuf,a0
		move.l		a0,usp
		move.l		sspbuf,-(sp)
		dc.w		_super
		addq.l		#4,sp

		rts

tline:
		move.l		x1,d0			;d0=X1
		move.l		x2,d1			;d1=X2
		move.l		y1,d2			;d2=Y1
		move.l		y2,d3			;d3=Y2
		move.l		d0,d6			;d6=temp.X1
		move.l		d2,d7			;d7=temp.Y1
		sub.l		d0,d1			;X2-X1
		move.l		d1,deltax		;delta X=work(deltax)

		sub.l		d2,d3			;Y2-Y1
		move.l		d3,deltay		;delta Y=work(deltay)

		cmp.l		#0,d1			;d1=0 ?
		bgt.l		plusx			;d1>0 then plusx
		beq.l		zerox			;d1=0 then zerox
		move.l		#-1,sgnx		;work(sgnx)=-1
		muls		#-1,d1			;d1=d1*-1
```

```
                bra             lcont1
zerox           move.l          #0,sgny                 ;work(sgnx)=0
                bra             lcont1
plusx           move.l          #1,sgnx
lcont1          move.l          d1,absdltx
                muls            #2,d1
                move.l          d1,absdltx2
                move.l          absdltx,d1
                cmp.l           #0,d3
                bgt.l           plusy
                beq.l           zeroy
                move.l          #-1,sgny
                muls            #-1,d3
                bra             lcont2
zeroy           move.l          #0,sgny
                bra             lcont2
plusy           move.l          #1,sgny
lcont2          move.l          d3,absdlty
                muls            #2,d3
                move.l          d3,absdlty2
                move.l          absdlty,d3
                cmp.l           d1,d3
                bgt.l           bigy
bigx            move.l          absdltx,d4
                muls            #-1,d4
                clr.l           d5
lloop1          move.l          d6,sx
                move.l          d7.sy
                bsr             psetxy
                add.l           sgnx,d6
                add.l           absdlty2,d4
                blt.l           contbigx
                add.l           sgny,d7
                sub.l           absdltx2,d4
contbigx        addq.l          #1,d5
                cmp.l           d1,d5
                blt.l           lloop1
                bra             lcont3

bigy            move.l          absdlty,d4
                muls            #-1,d4
                clr.l           d5
lloop2          move.l          d6,sx
                move.l          d7,sy
```

```
                bsr             psetxy
                add.l           sgny,d7
                add.l           absdltx2,d4
                blt.l           contbigy
                add.l           sgnx,d6
                sub.l           absdlty2,d4
contbigy        addq.l          #1,d5
                cmp.l           d3,d5
                blt.l           lloop2

lcont3          rts

psetxy:
                move.l          d6,tempx
                move.l          d7,tempy
                mulu            #$80,d7
                divu            #16,d6
                mulu            #2,d6
                add.l           d6,d7
                move.l          d7,adrsl
                mulu            #8,d6
                move.l          tempx,d7
                sub.l           d6,d7
                move.l          #$8000,d6
                lsr.l           d7,d6
                move.l          adrsl,d7
                add.l           #$E00000,d7

                move.l          d7,a4
                or.w            d6,(a4)

                add.l           #$20000,d7
                move.l          d7,a4
                or.w            d6,(a4)
                move.l          tempy,d7
                move.l          tempx,d6

                rts

*============== WORK AREA ============================

adrs1           dc.l            0
adrs2           dc.l            0
tempx           dc.l            0
```

```
tempy           dc.l            0
x1              dc.l            0
y1              dc.l            0
x2              dc.l            0
y2              dc.l            0
sx              dc.l            0
sy              dc.l            0
deltax          dc.l            0
deltay          dc.l            0
sgnx            dc.l            0
sgny            dc.l            0
absdltx         dc.l            0
absdlty         dc.l            0
absdltx2        dc.l            0
absdlty2        dc.l            0



                even
ret_param       dc.w    0
                dc.l    0
                ds.l    1

msg_error       dc.b    "Error",0

sspbuf          dc.l    0
uspbuf          dc.l    0

number          dc.l    0
max_len         dc.l    0
str_pointer     dc.l    0
ms_x            dc.l    0
ms_y            dc.l    0
pointer         dc.l    0
x_pos           dc.l    0
y_pos           dc.l    0
xs              dc.l    0
xe              dc.l    0
ys              dc.l    0
ye              dc.l    0
mous_x          dc.l    0
mous_y          dc.l    0
answer          dc.l    0
stack1          dc.l    0
```

```
stack2          dc.l    0
stack3          dc.l    0
temp_x_pos      dc.l    0
temp_y_pos      dc.l    0
max_x           dc.l    0
max_y           dc.l    0
spc_code        dc.b    $20,0

wxe1            dc.w    0
wye1            dc.w    0
wxe2            dc.w    0
wye2            dc.w    0
wxs1            dc.w    0
wys1            dc.w    0
wxs2            dc.w    0
wys2            dc.w    0

vram_buf1       dc.w    0
                dc.w    0
                ds.b    4096

vram_buf2       dc.w    0
                dc.w    0
                ds.b    4096

                end
```

さて，この外部関数はこれまでの総決算的なものです。かなりの数のIOCSコールを使用し，DDA法のラインルーチンまで使用しています。ソースプログラムもそれなりに長いものですが，これからの視覚に訴える新しい入力方法ではないでしょうか。表示したいメニューの項目を配列にセットし，指定した項目数と表示位置を設定するだけで，マウスにより選択したメニューの番号が返ってきます。なお，この関数を実行しているときは，以前にテキスト画面上に書かれていた文字をバッファに読み込み記憶してありますから，pop_up関数終了後は，元の画面に復帰しています。

pop_up関数を使用したサンプルプログラムは**第５－７図**の通りです。

```
1000 int x,y,n,bl,br
1010 dim str st(5)
1020 st(0)="X-BASIC"
1030 st(1)="X68000"
```

```
1040 st(2)="Human68k"
1050 st(3)="マイコン"
1060 st(4)="function"
1070 st(5)="end"
1080 mouse(0)
1090 mouse(1)
1100 msarea(0,0,767,511)
1110 repeat
1120      button()
1130      mspos(x,y)
1140      n=pop_up(st,6,x,y)
1150      print n
1160      button()
1170 until n=6
1180 mouse(0)
1190 end
1200 func button()
1210      while bl=0
1220           msstat(x,y,bl,br)
1230      endwhile
1240      while bl=-1
1250           msstat(x,y,bl,br)
1260      endwhile
1270 endfunc
```

第5－7図　POP_UPのサンプルプログラム

この第5－7図のプログラムは，pop_upの動作を確認するもので，マウスを左クリックするとポップアップメニューが現れます。この時に画面をクリアしてなければ，テキスト画面上に表示されている文字（例えば，上記のリスト等）の上に現れて，次に左クリックするとマウスカーソルが示している項目（メニュー）が選ばれて，その番号が返ってきます。ポップアップメニューの外で左クリックすると0が返ってきます。

これで外部定義関数の事例の紹介と解説を終わりますが，次の**第5－8図**のリストはこれまでに紹介した外部定義関数を使用して作られているテキスト画面のお絵かきソフトです。第1章から解説してきた内容もギッシリと詰ったものです。よく理解してX-BASICの奥の深さと秘めた可

能性について考えてみて下さい。

```
 10 /*
 20 int x,y,mx,my,count,bl,br
 30 dim int xx(2000),yy(2000)
 40 dim str a(4)={ "Copy(大)","copy(小)",
 50                  "All clear","End" }
 60 screen 2,0,1,1
 70 console 0,31,0
 80 mouse(0)
 90 mouse(1)
100 mouse(4)
110 mcset()
120 mscursor(1)
130 while 1
140     msstat(x,y.bl,br)
150     if bl=-1 then draw():continue
160     if br=-1 then pop()
170     count=0
180     mspos(mx,my)
190     xx(count)=mx
200     yy(count)=my
210 endwhile
220 end
230 func draw()
240     count=count+1
250     mspos(mx,my)
260     xx(count)=mx
270     yy(count)=my
280     tline(xx(count-1),yy(count-1),xx(count),yy(count),0)
290     tline(xx(count-1),yy(count-1),xx(count),yy(count),1)
300 endfunc
310 func pop()
320     int x,y,br,bl,v,w
330     mscursor(0)
```

```
340      msarea(2,2,760,510)
350      mspos(x,y)
360      v=pop_up(a,4,x,y)
370      switch v
380          case 1:mscursor(2):h_copy(1):break
390          case 2:mscursor(2):h_copy(0):break
400          case 3:tcls(-1):break
410          case 4:ending():break
420      endswitch
430      msarea(0,0,767,511)
440      mscursor(1)
450      repeat
460         msstat(x,y,bl,br)
470      until bl=0
480 endfunc
490 func ending()
500      tcls(-1)
510      console 0,31,1
520      print chr$(4)
530      mouse(0)
540      end
550 endfunc
560 func mcset()
570 dim int pat(32)={
580                     &B0001111111111111,
590                     &B0110011111111111,
600                     &B0111100111111111,
610                     &B1010011011111111,
620                     &B1010011101111111,
630                     &B1101111110111111,
640                     &B1101111111011111,
650                     &B1110111111101111,
660                     &B1111011111110111,
670                     &B1111101111111011,
```

```
680                     &B1111110111111101,
690                     &B1111111011111101,
700                     &B1111111101111011,
710                     &B1111111110110111,
720                     &B1111111111001111,
730                     &B1111111111111111,
740 /*
750                     &B0000000000000000,
760                     &B0110000000000000,
770                     &B0111100000000000,
780                     &B0010011000000000,
790                     &B0010011100000000,
800                     &B0001111110000000,
810                     &B0001111111000000,
820                     &B0000111111100000,
830                     &B0000011111110000,
840                     &B0000001111111000,
850                     &B0000000111111100,
860                     &B0000000011111100,
870                     &B0000000001111000,
880                     &B0000000000110000,
890                     &B0000000000000000 }
900 mscset(1,0,0,pat)
910 endfunc
```

第5－8図　おまけのTEXT_DRAW.BAS

プログラムの使用方法

RUNによりプログラム実行後，マウスをそのまま移動するとえんぴつカーソルもそれにつれて移動します。線を引きたいときは，左のボタンでドラッグしているあいだ線を書くことが出来ます。右ボタンをクリックするとポップアップメニューが現れ，画面のハードコピーか，画面クリアか，プログラム終了かを聞いてきます。キャンセルする時は，枠の外でボタンを押して下さい。なお，テキスト画面のハードコピーを実行させる外部定義関数（**第5－9図**）をおまけとして付け加えてありますから，**第5－8図**のサンプルプログラムを実行する前に、basic.cnfの中に定義しておいて下さい。

```
****************************************
*        Hard copy for X68000
*    presented by CZ masters club
****************************************

char_val        equ     $0004
int_ret         equ     $8001

adr_init        dc.l    return
adr_run         dc.l    return
adr_end         dc.l    return
adr_system      dc.l    return
adr_break       dc.l    return
adr_input       dc.l    return
adr_reserve1    dc.l    return
adr_reserve2    dc.l    return
adr_token       dc.l    token_table
adr_parameter   dc.l    param_adr_table
adr_exec        dc.l    exec_table
reserve         dc.l    0,0,0,0,0


token_table     dc.b    "h_copy",0,0

                even

param_adr_table dc.l    param_table

param_table     dc.w    char_val,int_ret


                .even

exec_table      dc.l    h_copy

                .even

h_copy:         move.l  12(sp),d0
                trap    #12             ;これは、コピーキー処理を行うものです

                clr.l   d0
                lea.l   ret_param(pc),a0
                lea.l   msg_error(pc),a1
```

```
return          rts

                .even
ret_param       dc.w    0
                dc.l    0
                dc.l    1

msg_error       dc.b    "Error",0

                .end
```

第5－9図　おまけの外部定義関数

ワンポイントテクニック　行番号

行番号は本文中にもあるように1から65535までの整数が使用出来ます。プログラムは，この行番号にそって行番号の小さい順に実行していきます。制御構造命令等で，行番号を無視した順序でプログラムを進めることが出来ます。また，行番号はダイレクト命令か，プログラムかを解釈するための大切なものです。隣り合う行番号に割り当てた数値が1，2，3……と並んでいると整然としていて美しく見えますが，もし，プログラムの変更等があり行間に別な行を挿入しなければならなくなっても，これでは新しい行の挿入ができないので行番号はできるだけ10，20，30……として下さい。ここへ15とか25という行番号を追加することができるようになった訳です。行番号を追加した後は，RENUMコマンドで行番号を整理しておくのも忘れないで下さい。「これでよし」と言うまでは，行番号はどんなにバラバラでも構わないのですから……。

それからRENUMで注意しなければならないことに，GOTO，GOSUBの飛先番地については新しい行番号を割り当ててくれません（下図参照）。

RENUMする前	RENUM後
10　INT A,B 15　PRINT A,B 20　A＝A＋1：B＝B＋2 30　GOTO 15	10　INT A,B 20　PRINT A,B 30　A＝A＋1：B＝B＋2 40　GOTO　15……飛先が変更されていない

そんな理由からもこのGOTO,GOSUBの使用はやめた方がいいでしょう。

究極の16ビット機 完全攻略本!!

第1部　X68000のすべて

第1章　X68000の魅力を探る

第2部　ビジュアルシェル入門

第2章　起動と停止
第3章　システムのバックアップとデータディスクの作成
第4章　ディスクウィンドウのオープン/クローズ
第5章　ファイルの活用
第6章　ディスクアイコンとディスクウィンドウ
第7章　アイコンの基本操作―移動，コピー，削除
第8章　コマンドメニューの活用
第9章　フォルダーの活用

第3部　コマンドシェル入門

第10章　コマンドシェル専用ディスクの作成
第11章　コマンドシェルの基礎
第12章　ディスクとファイルの基本オペレーション
第13章　ファイル基本オペレーション
第14章　UNIXライクな機能〈1〉―階層ディレクトリ
第15章　UNIXライクな機能〈2〉―リダイレクトとパイプライン
第16章　バッチ処理

第4部　システムの構築&各種テクニック

第17章　日本語フロントプロセッサの組み込み
第18章　RAMディスク＆SRAMディスク
第19章　福袋(贈物)の活用―マクロアセンブラ&リンカ
第20章　ビジュアルシェルでパラメータをつけて実行する方法
第21章　X-BASICプログラムテクニック―スプライトの活用法

資料編

資料1　X68000メモリマップ
資料2　X68000 I/Oポートアドレス一覧
資料3　テキストVRAMのメモリマップ
資料4　テキスト仮想画面のアドレス配置
資料5　テキストパレットアドレス
資料6　グラフィックVRAMメモリマップ
資料7　マクロアセンブラASの起動スイッチ
資料8　マクロアセンブラASの書式一覧
資料9　マクロアセンブラASのエラーメッセージ
資料10　リンカLKの起動スイッチ
資料11　スプライトパターン＆カラーコードエディタの使用法
資料12　ファンクションコールとIOCS
資料13　ファンクションコールの使い方
資料14　IOCSの使い方
資料15　IOCS一覧
資料16　68000インストラクションセット
資料17　X68000回路図

月刊マイコン別冊

OS入門から各種テクニック・内部解析まで

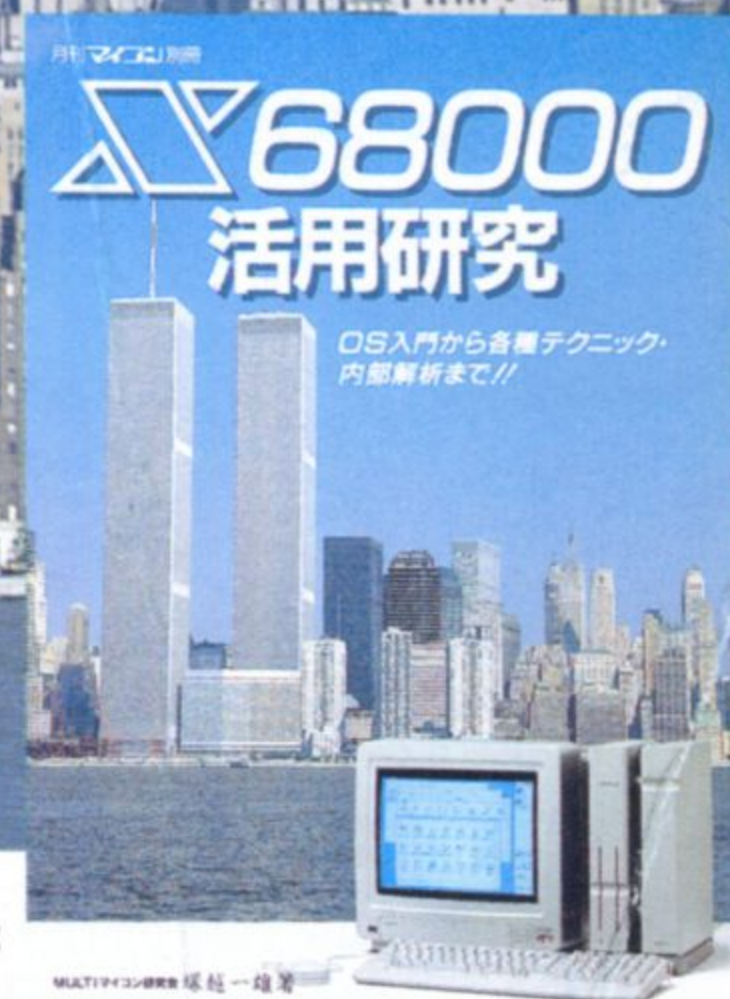
X68000 活用研究

MULTIマイコン研究会　塚越一雄 著

B5判364頁 定価2,200円(〒300円)

好評発売中!!

電波新聞社　出版販売部

東京本社　〒141　東京都品川区東五反田1-11-15　☎(03)445-6111
大阪本社　〒530　大阪市北区中之島3-2-4 朝日新聞ビル　☎(06)203-3361
西部本社　〒812　福岡市博多区博多駅前2-13-23扇寿ビル　☎(092)431-7411

SHARP
New-Life People

あふれるクリエイティブマインド——NEW Z-BASIC搭載。

ADVANCED TURBO新登場。

NEW Z-BASIC搭載

多色グラフィック、カラー画像デジタイズ、ステレオFM音源、バンクメモリ対応などクリエイティブワークを強力にサポートするAV指向の高水準BASICです。グラフィック用関数、X68000と命令コンパチの拡張MMLをはじめ使い込むほどに凄さがわかるパワフルなBASICを搭載しました。

先駆のアート機能

量子化、モザイク、反転などトリック取り込み処理をサポートしたカラー画像デジタイズ機能標準装備。さらに、クロマキー合成、インターレーススーパーインポーズ、4,096色対応ニューテロッパ機能、8重和音のステレオFM音源。先駆のZアビリティがパソコンクリエイターを魅了します。●メインメモリ128KB標準実装（NEW Z-BASICで最大576Kバイトまでサポート）した大容量設計●1Mバイトフロッピー2基搭載●JIS第1/第2水準漢字、「システム・ユーザー辞書」標準装備●簡単操作のマウス標準装備●X1ターボシリーズの豊富なソフト資産が活用できるコンパチブル設計●多彩な通信ツール*のサポートでパソコン通信に対応●ドットピッチ0.31mmの高精細カラーディスプレイテレビ* ※別売

X1 パソコンテレビ turbo ZII

品名	型番		価格
パーソナルコンピュータ＋キーボード	CZ-881C-BK(ブラック)	標準価格	179,800円
14型カラーディスプレイテレビ	CZ-880D-BK(ブラック)	標準価格	109,800円
14型カラーディスプレイテレビ	CZ-830D-BK(ブラック)	標準価格	98,000円
チルトスタンド	CZ-6STI-B(ブラック)	標準価格	5,800円

※写真のディスプレイはCZ880Dです。

シャープ株式会社
●お問い合わせは…シャープ㈱電子機器事業本部システム機器営業部〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)
電子機器事業本部テレビ事業部第4商品企画部〒162 東京都新宿区市谷八幡町8番地☎(03)260-1161(大代表)

資料請求券
X1 turbo ZII
X68000活用研究II

電波新聞社 〒141 東京都品川区東五反田1－11－15 電話 (03)445－6111(大代) 定価2,000円 雑誌68469-46

マイコン別冊 X68000活用研究II 昭和六十三年五月二十日第二刷発行

マイコン別冊
X68000活用研究II
宮原哲也／深沢幸二共著
電波新聞社